Pearson Education

ADDISON-WESLEY

Flash 5 ActionScript

Pearson Education

ADDISON-WESLEY

Pearson Education

ADDISON-WESLEY

Jan Brücher
Marc Hugo

Flash 5 ActionScript

Professionelle Praxislösungen für Programmierer und Webdesigner

Bitte beachten Sie: Der originalen Printversion liegt eine CD-ROM bei.
In der vorliegenden elektronischen Version ist die Lieferung einer CD-ROM nicht enthalten.
Alle Hinweise und alle Verweise auf die CD-ROM sind ungültig.

ADDISON-WESLEY

An imprint of Pearson Education

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

Pearson Education

ADDISON-WESLEY

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Ein Titeldatensatz für diese Publikation ist bei der Deutschen Bibliothek erhältlich

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

10 9 8 7 6 5 4 3 2 1

04 03 02 01

ISBN 3-8273-1763-0

Einbandgestaltung:	Helmut Kraus, Düsseldorf
Lektorat:	Klaus Hofmann, khofmann@pearson.de
Korrektorat:	Angelika Obermayr, tech.doc
Herstellung:	Anna Plenk, aplenk@pearson.de
Satz:	mediaService – Dirk Matzke, Siegen (www.mediaproject.net)
Druck und Verarbeitung:	Media-Print, Paderborn

Printed in Germany

Pearson Education

ADDISON-WESLEY

Inhaltsverzeichnis

Vorwort

Sehr geehrte Leserin, sehr geehrter Leser,

die Skriptsprache ActionScript erweitert die Funktionen, die Ihnen Flash bei der Gestaltung von interaktiven und animierten Webangeboten bietet, beträchtlich. Die Programmversion 5 stellt gerade im Hinblick auf ActionScript noch einmal eine wesentliche Weiterentwicklung dar. Es erschließen sich so ganz neue Anwendungen, so etwa bei der Programmierung von Spielen. Allerdings ist der Umgang mit dem Programm und der zugehörigen Skriptsprache mittlerweile vergleichsweise komplex und damit erklärungsbedürftig.

Wir verfolgen mit diesem Buch insofern ein zweifaches Anliegen: Zum einen wollen wir Ihnen einen guten Einstieg in ActionScript geben, also die Herangehensweise, die Neuerungen und die wesentlichen Befehle erklären. Zum anderen wollen wir Ihnen anhand zahlreicher Praxisbeispiele zeigen, was mit ActionScript alles möglich geworden ist. Diese fortgeschrittenen Anwendungen haben wir größtenteils selbst entwickelt. Sie können sie Schritt für Schritt nachvollziehen und dank der beigegebenen CD-ROM auch selbst einsetzen und weiterentwickeln.

Wie wir aus eigener Erfahrung und der Beteiligung in den einschlägigen Flash-Foren wissen, bleiben bei der Arbeit mit ActionScript nicht selten Fragen offen. Deshalb finden Sie in diesem Buch nicht zuletzt auch viele Tipps und workarounds sowie einen kommentierten Anhang zu den wesentlichen ActionScript-Befehlen. Auf diese Weise wird Ihnen dieses Buch eine wichtige Hilfe für die tägliche Arbeit sein und auch als Nachschlagewerk gute Dienste leisten.

An dieser Stelle möchten wir uns auch noch einmal beim Lektoratsteam, vor allem bei Herrn Hofmann, für die ausgezeichnete Zusammenarbeit, Geduld und zügige Realisierung dieses Buches bedanken. Spezieller Dank geht des weiteren an Frank Wehrsenger für die Unterstützung und konstruktive Kritik am Buch.

Viel Spaß bei der Arbeit mit diesem Buch

Jan Brücher, Marc Hugo Februar 2001

Pearson Education

ADDISON-WESLEY

1 Einführung

Macromedia Flash 5 ist ein mächtiges Tool zur Erstellung von Internetapplikationen, Web-Oberflächen oder Interaktiver CD-ROM. Ein wesentlicher Vorteil ist hierbei, dass im Gegensatz zu HTML-Oberflächen Flash-Filme auf allen Browsern und plattformunabhängig das gleiche Erscheinungsbild erzeugen. Wer täglich Internetapplikationen erstellt, weiß dies sehr zu schätzen, vor allem, da hier (fast) keine gestalterischen Grenzen gesetzt sind. Allerdings wird immer wieder die Notwendigkeit des Flash-Plugins kritisch angemerkt. Dazu ist folgendes zu sagen:

- Die meisten Browser haben dieses Plugin direkt integriert.
- Das Plugin wird, wenn es nicht schon im Browser integriert ist, nur einmal heruntergeladen.
- Das Flash-Plugin ist für die gängigsten Plattformen erhältlich.

Im Folgenden zeigen wir Beispiele und Workarounds sowohl für Flash 4 als auch für Flash 5. Flash 5-Features werden nur von dem aktuellen Flash-Plugin unterstützt: Das bekannteste Beispiele ist wohl `Mouse.hide()` (Mauscursor verbergen), allerdings gibt es die Möglichkeit, die neue Funktion `_xmouse` nach Flash 4 zu exportieren. Der entsprechende Workaround für Flash 4 wird an späterer Stelle noch vorgestellt.

1.1 Flashbasics

Um alle folgenden Tipps und Anleitungen nachvollziehen zu können, müssen Sie über einige Grundkenntnisse zu Flash verfügen.

Die Werkzeuge

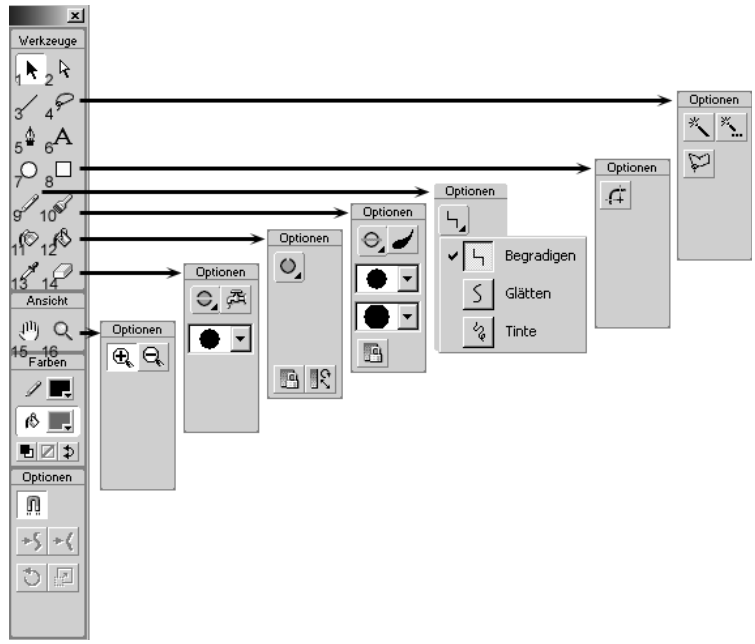
Im Folgenden gehen wir zunächst auf die verschiedenen Werkzeuge und ihre Funktion ein.

Zu den einzelnen Werkzeugen:

1. Das Pfeilwerkzeug (Hotkey V) zur Auswahl einzelner Objekte, Objektsegmente oder Objektgruppen: Bei gedrückter ⇧-Taste können mehrere Objekte selektiert werden. *Hotkey V*
2. Das Unterauswahlwerkzeug (Hokey A) zur Bearbeitung einzelner Pfade, bzw. Spline-Objekte. *Hotkey A*

Hotkey N ③ Das Linienwerkzeug (Hotkey N) zeichnet Linien, bei gedrückter ⇧-Taste jeweils im Winkel von 45° bzw. eines Mehrfachen davon.

Hotkey L ④ Das Lassowerkzeug (Hotkey L) wählt Bereiche frei aus; weitere Optionen sind der Zauberstab, der eingestellte Farbabgrenzungen berücksichtigt, und der Polygon-Modus, in dem alle Punkte des Auswahlbereiches durch Linien verbunden sind.



Hotkey P ⑤ Das Stiftwerkzeug (Hotkey P), mit dem einzelne Knoten der Pfade eingezeichnet werden können. Bei nochmaligem Anklicken der Knoten werden diese gelöscht.

Hotkey T ⑥ Das Textwerkzeug (Hotkey T) zur Erstellung dynamischer oder statischer Textfelder. Diese können später sowohl zur dynamischen Eingabe oder Ausgabe genutzt werden, aber auch um statischen Text darzustellen.

Hotkey O ⑦ Das Ellipsenwerkzeug (Hotkey O), mit dem auch besondere Ellipsen wie der Kreis gezeichnet werden können. In diesem Fall erscheint ein fetter Ring unter dem Cursor. Sie können dies erzwingen, indem Sie die ⇧-Taste gedrückt halten.

Hotkey R ⑧ Mit dem Rechteckwerkzeug (Hotkey R) können Sie je nach Option ein weniger rechtwinkliges Rechteck mit abgerundeten Kanten erzeugen.

Hotkey Y ⑨ Mit dem Freihandwerkzeug (Hotkey Y) können Sie Zeichnungen anfertigen, die je nach Option mehr oder weniger begradigt werden. Diese Pfade können Sie mit Hilfe des Unterauswahlwerkzeuges oder des Stiftwerkzeuges korrigieren.

Hotkey B ⑩ Das Pinselwerkzeug (Hotkey B) ist zum Freihandzeichnen am besten geeignet. Für Besitzer eines Digitizers (Grafiktablett) bietet sich hier auch die

Möglichkeit, den Stiftandruck mit in die Strichstärke einfließen zu lassen. Alle weiteren Optionen sind selbsterklärend.

⑪ Das Tintenfasswerkzeug (Hotkey S) dient dem nachträglichen Ändern der Pfadstärke sowie der Pfadfarbe. *Hotkey S*

⑫ Das Farbeimerwerkzeug (Hotkey K) dient dem nachträglichen Ändern der Füllungen oder der mit dem Pinselwerkzeug gezeichneten Objekte. Des Weiteren bietet es nun auch die Funktion, Füllungen nachträglich zu bearbeiten, was auch beinhaltet, deren Ausrichtung und Übergangsbereich frei definieren zu können. Möchten Sie ein gezeichnetes Pfad-Objekt mit einer Farbe bzw. Füllung versehen, können Sie optional die Lückentoleranz einstellen. *Hotkey K*

⑬ Die Pipette (Hotkey I) ist ein richtiges Multitalent. Mit ihr kann man Formate jeglicher Art (Text, Farbe/ Verlauf, Strichfarbe/-stärke) aus Objekten auslesen und anderen Objekten zuordnen. *Hotkey I*

⑭ Die Optionen des Radiergummiwerkzeugs (Hotkey E) sind selbsterklärend: Der Wasserhahn entspricht einer Radierung gleich der Farbfüllung mit dem Farbeimer. So können Sie ganze Bereiche radieren, ohne angrenzende Farbfüllungen zu beeinträchtigen. *Hotkey E*

⑮ Das Handwerkzeug (Hotkey H) ist eines der unspektakulärsten Tools der ganzen Palette, aber mit Sicherheit eines der meistbenutzten. Mit Hilfe der Hand können Sie per Drag and Drop ohne die Bildlaufleisten navigieren. Sie erhalten die Hand aber auch durch die gedrückt gehaltene Leertaste, wenn Sie z. B. Ihr Werkzeug gewählt habe und nur zur weiteren Bearbeitung navigieren müssen. Beim Loslassen der Leertaste wird wieder das letzte Werkzeug automatisch ausgewählt. Sie ersparen sich dadurch im späteren Workflow eine Menge Zeit und Nerven. *Hotkey H*

⑯ Das Vergrößerungswerkzeug (Hotkeys M,Z) ist geeignet, wenn man einen fertigen Flashfilm noch einmal kontrollieren möchte oder von Anfang an in sehr kleinen Bereichen arbeitet. Hier bietet es sich an, den zu vergrößernden Bereich mit gedrücktem Lupencursor zu markieren: So umgeht man die grobstufigen voreingestellten Zoomschrittweiten. *Hotkey M,Z*

Parallelen und Unterschiede: Flash 5 ActionScript – JavaScript

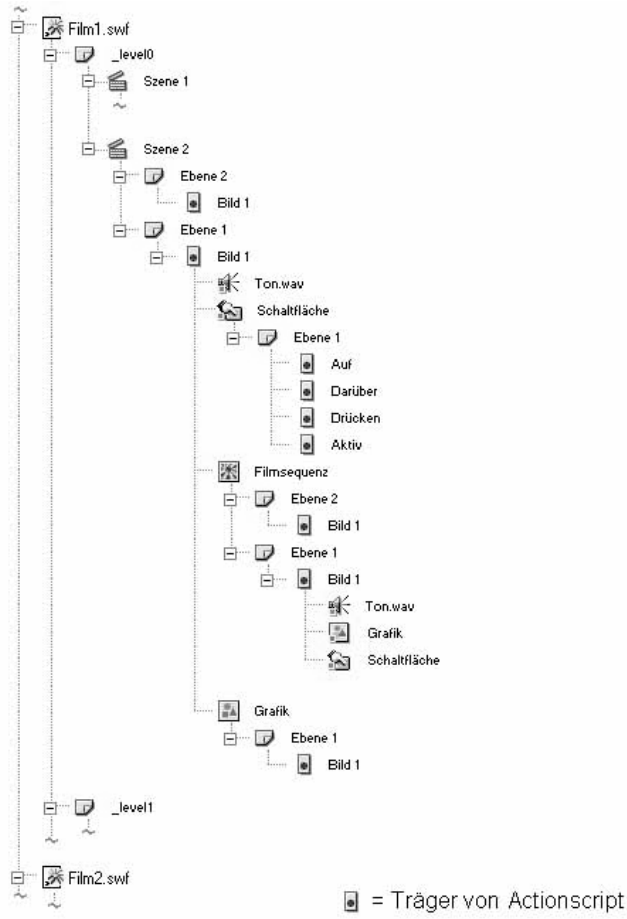
Flash 5 ActionScript lehnt sich als Nachfolger zu Flash 4 an die sprachlichen Konventionen von JavaScript an. Daher wurde die Syntax stark angeglichen – der Teufel liegt hier allerdings im Detail. Grundsätzlich ist der Ansatz aber vernünftig, denn so wird der Entwickler nicht mit einer komplett neuen Sprache konfrontiert.

- Flash unterstützt keine Browser-spezifischen Objekte wie document, window oder anchor.
- Flash verwandelt nicht zugewiesene Variablen (undefined) im Falle einer Zeichenkette zu „ “ anstelle zu „`undefined`“ wie es das JavaScript macht.

- Flash verwandelt nicht zugewiesene Variablen (`undefined`) im Falle eines Wertes zu `0` anstelle zu `NaN` (Not a Number, kein Wert) wie es JavaScript macht.
- Flash unterstützt Slash-Syntax, d.h. einzelne Unterobjekte werden wahlweise mit dem Slash (/) oder mit dem Punt (.) vom übergeordneten Objekt getrennt. JavaScript ist ausschließlich auf Punkt-Syntax beschränkt.

1.2 Flash 5 – Der Einstieg

In dem folgenden Abschnitt möchte ich Ihnen grundlegende Funktionen, Struktur und Aufbau von Flash erläutern. Die einzelnen beispielhaften Szenen und Filmsequenzen werden nacheinander abgearbeitet, wenn keine Sprungbefehle vorliegen.

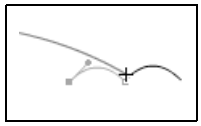


Anders als bei HTML lassen sich so Animationen erstellen und Effekte (z.B. Texteinblendungen) erzeugen.

Animation: Form-, Pfad- und Bewegungstweenings

Für eine einfache Animation eines fliegenden Vogels muss zuerst der Vogel als eigenständige Filmsequenz angelegt werden. Diese Filmsequenz beinhaltet die Animation des Flügelschlagens. Den Flug über den Horizont werden wir mittels einer Pfadanimation verwirklichen.

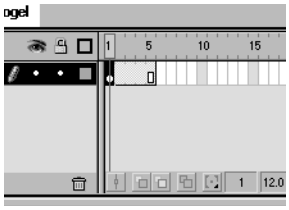
Das entsprechende Beispiel finden Sie unter dem Namen „Flug.fla“ auf der CD-ROM.



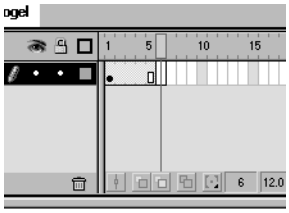
Zuerst wird der Vogel – oder besser die Möwe – angelegt. Hierzu soll für das Formtweening eine Strichanimation genügen. Zeichnen Sie eine Linie mit dem Linienwerkzeug (N) und krümmen Sie diese, um den ersten „Flügel“ zu erstellen. Wiederholen Sie dies nun für die andere Hälfte.

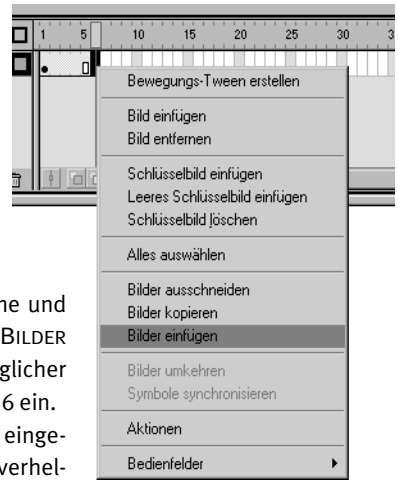
Nun müssen Sie zusätzliche Frames einfügen, um ein neues Schlüsselbild zu erstellen. Als Frame wird ein einzelnes Bildsegment aus der Zeitleiste bezeichnet – quasi die kleinste Zeiteinheit in Flash. Ein Schlüsselbild bezeichnet einen Frame, in dem alle für ein Tweening maßgeblichen Objektpositionen und Eigenschaften festgelegt sind.

Markieren Sie den aktuellen Frame in der Zeitleiste und drücken Sie F5 für jeden neuen Frame, in diesem Fall also vier mal.

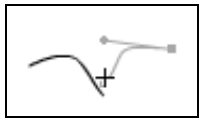


Erstellen Sie nun mittels Kopieren des ersten Frames ein neues Schlüsselbild. Erstellen Sie auf Frame 6 dafür zunächst ein leeres Schlüsselbild. Markieren Sie den entsprechenden Frame und drücken Sie F7.





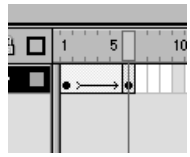
Nun markieren Sie den ersten Frame und kopieren per rechtem Mausklick „BILDER KOPIEREN“ die „Möwe“ in ursprünglicher Form. Fügen Sie diese nun in Frame 6 ein.



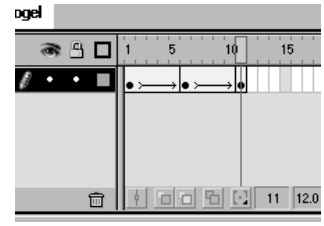
Der nun neu eingefügten Möwe verhelfen Sie per Unterauswahlwerkzeug zum Flügelschlag, mittels der Pfade verändern Sie die Möwe Ihren Wünschen entsprechend. In meinem Beispiel hebt diese nur die Flügel.

Markiert man nun die Frames 1 bis 5 und wählt FORM als Tweening unter Bild, erzeugt Flash automatisch die Zwischenschritte der Animation. Die Option FORM beschreibt hier nur die Art der Zwischenbilder. Beim Formtweening wird das Schlüsselbild zum nächsten Schlüsselbild, in das getweent wird, gewarpt, d.h. nur die Form verändert.

Diese Tweenings sind in der Zeitlinie hellgrün und mit einem Pfeil gekennzeichnet.

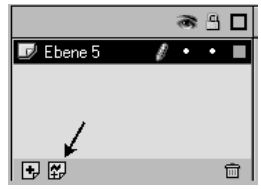


Wenn alles gut gegangen ist, erhalten Sie nebenstehendes Bild. Um die Möwe fertig zu stellen, müssen wir die Flügel wieder zurückführen, dies geht mittels ein paar kurzer Eingaben. Fügen Sie hinter den sechsten Frame wieder vier weitere Frames ein; in dem elften erstellen Sie mittels F7 ein leeres Schlüsselbild. Kopieren Sie das erste Bild und fügen Sie es in das neu erstellte leere Schlüsselbild ein. Markieren Sie die Frames 6 bis 10 und wählen Sie wieder unter BILD › TWEENING › FORM. Der fertige Film müsste nun folgende Timeline haben:



Da sich alle Filme, die auf die Hauptbühne gebracht werden (sofern nicht eingeschränkt), automatisch wiederholen, sind wir nun mit der Möwe fertig und können uns nun ihrer Flugbahn über dem Horizont widmen.

Dazu verwenden wir ein Pfadtweening. Fügen Sie zunächst die Möwe aus der Bibliothek bzw. Library auf der Hauptbühne ein (in der Bibliothek befinden sich alle verwendeten Instanzen, Grafiken und Sounds, ob aktiv oder inaktiv). Um dieser ihren Weg zum Horizont zu weisen, benötigen wir eine Pfadebene. Diese wird nur den einen Pfad beinhalten, der den Weg der Möwe beschreibt.

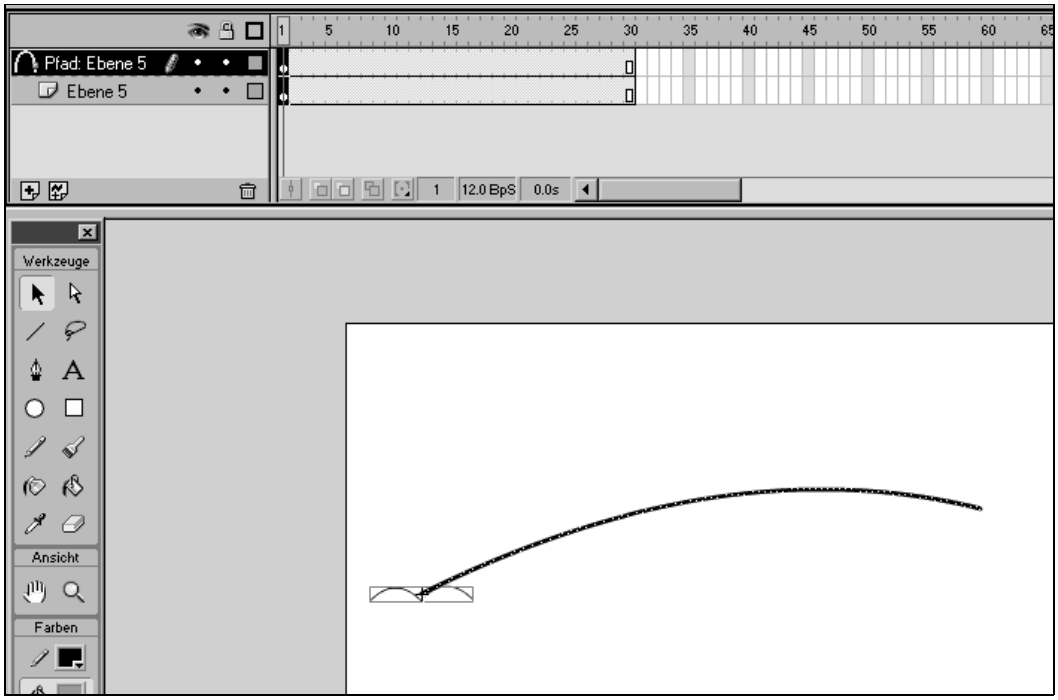


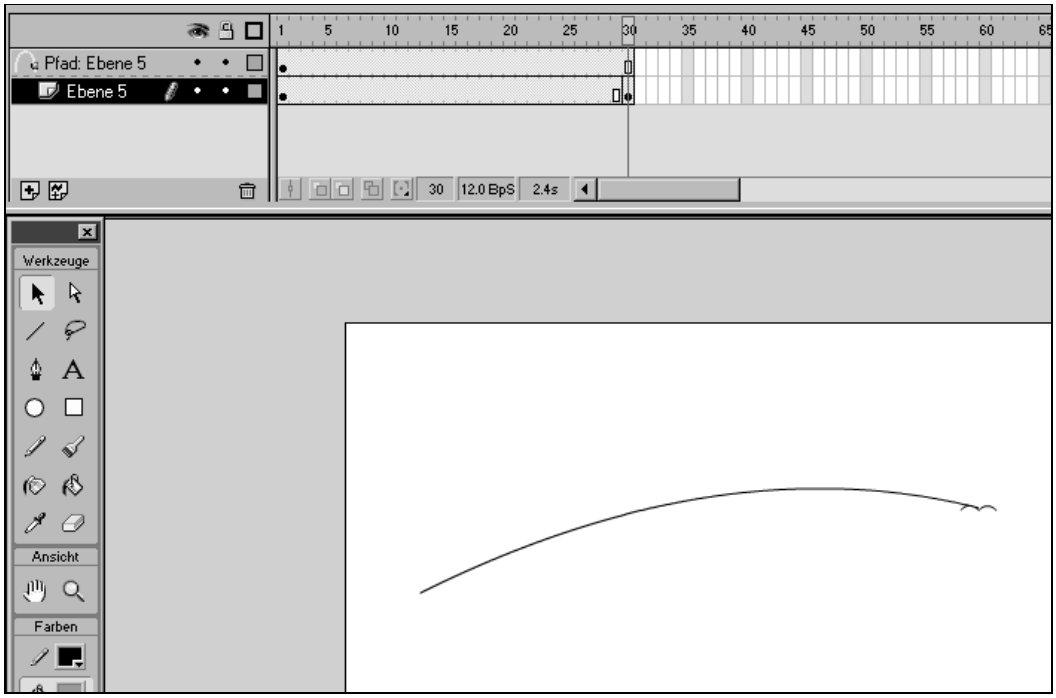
Die Pfadebene erzeugen Sie hier:

Legen Sie nun einen Pfad mit der gewünschte Flugbahn fest. Der Pfad sollte sich auf der Pfadebene befinden.

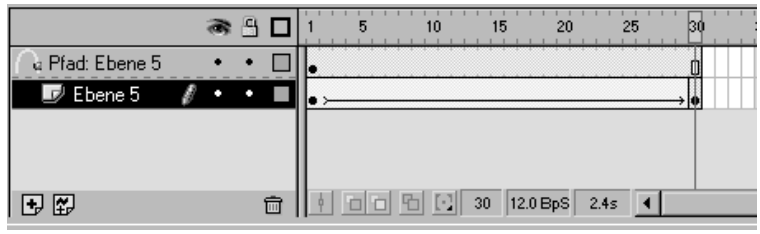
Markieren Sie nun den ersten Frame beider übereinander liegender Ebenen. Fügen Sie nun, je nach Animationsdauer, entsprechend viele Frames ein F5. In dem Beispiel reichen insgesamt 30 Frames.

Kopieren Sie nun den ersten Frame der unteren Ebene in den Frame 30; es wird automatisch ein neues Schlüsselbild erzeugt. Platzieren Sie die Möwe im ersten Frame an ihrer Startposition, im letzten an die Zielposition. Tiefe wird dadurch erzeugt, dass die Möwe am Horizont kleiner wird.


Pfad: Ebene 5
Ebene 5
1
5
10
15
20
25
30
35
40
45
50
55
60
65
1
12.0 BpS
0.0s
Werkzeuge
Ansicht
Farben


Pfad: Ebene 5
Ebene 5
1
5
10
15
20
25
30
35
40
45
50
55
60
65
30
12.0 BpS
2.4s
Werkzeuge
Ansicht
Farben

Markieren Sie nun in der unteren Ebene die Frames 1 bis 29 und wählen Bild › Tweening › Bewegung an. Die Zeitlinie müsste nun wie folgt aussehen:



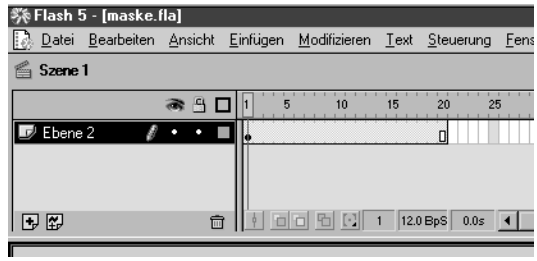
Der Pfeil ist im Gegensatz zum Formtweening blau unterlegt.

Fügen Sie noch eine Ebene mit einem passenden Hintergrund ein – fertig ist ihre Animation.

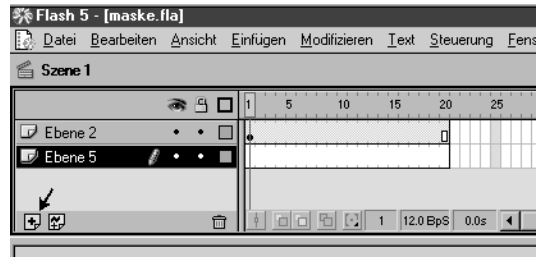
Ebenenmasken

Nachdem Sie erfahren haben, wie man Bewegungen in Flash umsetzt, werden wir näher auf Ebenenmasken eingehen. Um den wahrscheinlich bekanntesten Maskeneffekt, den Spotlighteffekt, zu erzeugen, legen Sie zunächst einen Text mit beliebigem Inhalt im ersten Frame des Filmes an.

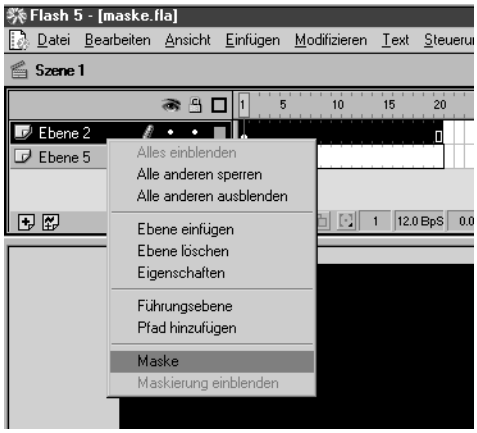
Ergänzen Sie nun den Film mittels F5 um 19 weitere Frames, so dass Sie nun insgesamt 20 Frames haben.



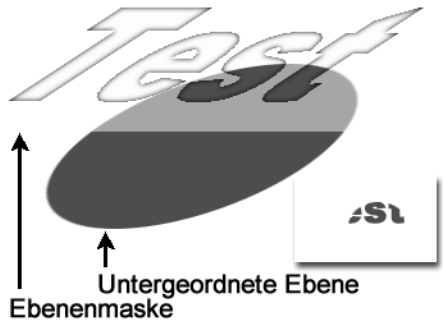
Danach legen Sie eine neue Ebene **unter der aktuellen** an.



Nun können Sie die obere der beiden Ebenen in eine Ebenenmaske verwandeln. Per rechtem Mausklick erhalten Sie ein Kontextmenü, aus dem Sie die Option MASKE auswählen.

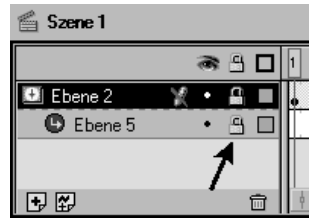


Die untergeordnete Ebene wird miteinbezogen und ist nur an den Stellen sichtbar, an denen sich Objekte auf der darüber liegenden Ebene befinden:



Deaktivieren Sie den Schreibschutz der Ebenen wieder durch einfaches Anklicken der Schloss-Symbole.

Nun erstellen Sie in der „maskierten" Ebene das Tweening eines gelben Kreises, der in den Frames 1 bis 10 vom Anfang bis zum Ende des Schriftzuges wandert und dann in den Frames 11 bis 20 wieder zur Ausgangsposition zurückkehrt.



In der Zeitleiste müsste es dann so aussehen:

Maske.fla

Sofern Sie den Ebenenschreibschutz nicht aktiviert haben, sehen Sie nun sowohl die Maske als auch die darunter liegende Ebene.

Weitere Effekte können Sie durch eine leichte Ergänzung des bisher Erarbeiteten hinzufügen: Kopieren Sie den Schriftzug, fügen Sie ihn genau (Strg + ⇧ + V) unter die Maske in eine neue, untergeordneten Ebene wieder ein. Wenn Sie nun für den Schriftzug anstelle des hellen Gelbs des Kreises ein etwas abgedunkelteres Gelb wählen, hat das den Effekt, als leuchte man in der Dämmerung umher.

Wenn Sie mit dem Programm auf Macintosh arbeiten, sollten Sie an Stelle der Strg - die Q -Taste betätigen.

Die Nachtversion sehen Sie in der folgenden Abbildung:

Und hier die Dämmerversion:

1.3 Programmiergrundlagen

Falls Sie noch keine Programmierkenntnisse haben, gibt Ihnen das folgende Kapitel eine kurze Einführung.

Wie in jeder Programmiersprache bietet auch Flash ActionScript die Möglichkeit, mittels Befehlen Objekte zu verändern, Operationen auszuführen und Werte zu verarbeiten. Die Gesamtübersicht hierzu finden Sie im Index. Die Syntax von ActionScript wurde mit der Version 5 an JavaScript angeglichen und bietet nun die so genannte **Punkt-Syntax**.

Hierzu kurz alle erlaubten Schreibweisen:

```
_root.Instanz.Wert
_level0.Instanz.Wert
_level0/Instanz:Wert
```

Alle drei Ausdrücke geben den Wert einer Instanz zurück. Die Reihenfolge der Instanznamen entspricht der Hierarchie, d.h., wenn der ersten Instanz noch eine zweite untergeordnet ist, erreicht man deren Werte wie folgt:

```
_root.ersteInstanz.zweiteInstanz.Wert
```

Die **Instanz** selbst ist ein Objekt, d.h. eine Art Container, der alle möglichen Werte, Eigenschaften oder Methoden beinhalten kann. Ein Beispiel ist das im ActionScript enthaltene Objekt `Math`. Es beinhaltet Werte, in diesem Fall Konstanten, und Methoden, bzw. Funktionen wie `Sin()`, `Cos()` etc. Diese Funktionen können unabhängig von `_root` referenziert werden.

```
Math.sin(Winkel)
Math.PI
```

Die erste Operation gibt den Wert des Sinus des Winkels (im Bogenmaß) zurück, die zweite den Wert von π (= 3,14 ...).

Sie können natürlich auch eigene Funktionen erzeugen. Wenn Sie eine Filminstanz anlegen und diese die folgende Funktion beinhaltet,

```
function myFkt(Zahl){
Wert = Zahl * Zahl;
return Wert;
}
```

können Sie diese mit

```
_root.Instanz.myFkt(Eingabe);
```

ansprechen. In diesem Fall wird die Quadratzahl des Eingabewertes zurückgegeben.

Natürlich bietet Ihnen ActionScript die Möglichkeit, Objekten Konstanten und Variablen zuzuordnen. Dies erfolgt durch eine Konstanten- oder Variabelendeklaration in einem Frame des Movies.

Unter den **Eigenschaften eines Objektes** kann man in ActionScript folgende Werte abfragen:

`_alpha`	liefert die Tranzparenz der Instanz zurück bzw. setzt diese
`_currentframe`	aktuelle Framenummer
`_droptarget`	Rahmen/ Zielpfad eines Objektes, kann bei Kollisionsdetection verwendet werden
`_focusrect`	gibt an, ob aktive Texteingabefelder markiert werden
`_framesloaded`	liefert die aktuelle Anzahl der geladenen Frames zurück
`_height`	liefert die Höhe einer Instanz zurück bzw. stellt diese ein
`_highquality`	gibt die Einstellung der Wiedergabequalität zurück bzw. setzt diese
`_name`	gibt den festgelegten Namen für eine Instanz zurück bzw. setzt diesen
`_quality`	gibt die Einstellung der Wiedergabequalität zurück bzw. setzt diese
`_rotation`	liefert die Drehung einer Instanz zurück bzw. setzt diese
`_soundbuftime`	liefert die Größe des Sundpuffers (default=5) in Sekunden zurück, bzw setzt diesen
`_target`	liefert den Zielpfad der Instanz zurück
`_totalframes`	liefert die Anzahl aller Frames des Filmes zurück
`_url`	liefert die URL der .swf-Datei zurück
`_visible`	liefert die Sichtbarkeit einer Instanz zurück bzw. stellt diese ein
`_width`	liefert die Breite einer Instanz zurück bzw. stellt diese ein
`_x`	liefert die X-Koordinaten einer Instanz relativ zu deren Koordinatenursprung zurück bzw. stellt diese ein

_xmouse	liefert die X-Koordinaten des Mauscursors relativ zu deren Koordinatenursprung zurück
_xscale	liefert die X-Skalierung einer Instanz zurück bzw. stellt diese ein
_y	liefert die Y-Koordinaten einer Instanz relativ zu deren Koordinatenursprung zurück bzw. stellt diese ein
_ymouse	liefert die Y-Koordinaten des Mauscursors relativ zu deren Koordinatenursprung zurück
_yscale	liefert die Y-Skalierung einer Instanz zurück bzw. stellt diese ein

Wie zu erkennen ist, sind sämtliche Eigenschaften, die Flash kennt, durch einen vorangestellten Unterstrich (_) gekennzeichnet.

Um eine Interaktion zu erzeugen, sind aber auch immer sogenannte „Eventhandler“ nötig, die die Eingaben, bzw. Zustände laufend abfragen.

Um eine Instanz zu überwachen gibt es die `onClip(EVENT){}`-Abfrage. Folgende Ereignisse können dort verfolgt werden:

load	diese Bedingung ist erfüllt (true), wenn die Instanz geladen wird
enterFrame	diese Bedingung ist erfüllt, wenn die Instanz aufgerufen wird
unload	diese Bedingung ist erfüllt, wenn die Instanz entladen wird
mouseDown	diese Bedingung ist erfüllt, wenn eine Maustaste gedrückt wird
mouseUp	diese Bedingung ist erfüllt, wenn eine Maustaste losgelassen wird
mouseMove	diese Bedingung ist erfüllt, wenn die Maus bewegt wird
keyDown	diese Bedingung ist erfüllt, wenn eine Taste gedrückt wird
keyUp	diese Bedingung ist erfüllt, wenn eine Taste losgelassen wird
data	diese Bedingung ist erfüllt, wenn Daten per loadVariables bzw. loadMovie geladen werden

Instanzen, die als Buttons definiert sind, können wiederum andere Dinge „überwachen“. Hier verwenden Sie `on(EVENT){}`

dragOut	bei gedrückter Maustaste wird der Button verlassen
dragOver	bei gedrückter Maustaste wird der Button getroffen
keyPress "X"	es wird die Taste „X“ (beliebig wählbar) gedrückt
press	Button wird gedrückt
release	Maustaste wird über dem Button losgelassen
releaseOutside	Button wird gedrückt und die Maustaste wird außerhalb des Buttons losgelassen
rollOut	der Mauscursor verlässt den Button
rollOver	der Mauscursor überquert den Button

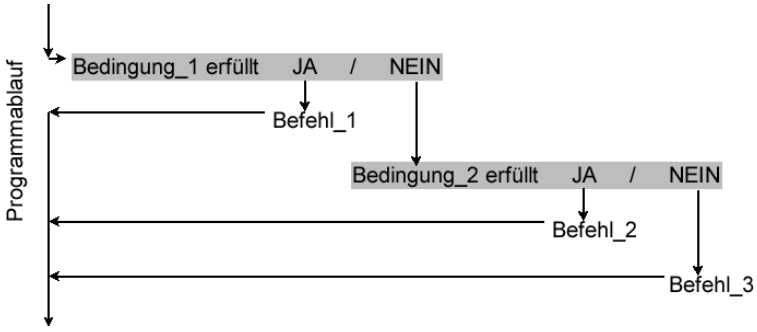
Wenn Sie Werte, seien es Zeichenketten, Zahlen oder Boolean (`true` oder `false` / Wahrheitsaussagen), in einem ActionScript verarbeiten wollen, so müssen Sie diese zwischenspeichern. Hierfür nimmt man Variablen, die noch nicht belegte Namen haben können. Es empfiehlt sich immer, diese durch einheitliche Namensgebung kenntlich zu machen – so ist ein Name als Variable gut als sName zu erkennen (s Bezeichnet hier String, Zeichenkette), Zahlen wie Alter z. B. in einer Variable iAlter (i Integer, ganzzahliger Ausdruck). Nun können Sie Ihre Ausdrücke verarbeiten ohne auf das Objekt zugreifen zu müssen. Sind die Werte unveränderlich, so spricht man hier von konstanten Ausdrücken oder kurz Konstanten.

▶ **Schleifen/Abfragen:**

Wenn Sie für eine weiterführende Programmierung Abfragen benötigen (Passwortabfrage), setzen Sie eine sogenannte **if-Abfrage ein**. Folgende Syntax wird verwendet:

```
if (Bedingung_1){
    Befehl_1;
  } else if (Bedingung2) {
    Befehl_2;
  } else {
    Befehl_3;
}
```

Folgender Aufruf erfolgt bei Abarbeitung des Scripts:



Ein weiteres wichtiges Element ist die **for-Schleife**:

```
for (i=1, i<10, i++){
duplicateMovieClip ( _root.Instanzname , "Test"+i, i);
}
```

Dies ist bei Flash eine typische Anwendung der for-Schleife. Hier wird der MovieClip „Instanzname“ neunmal dupliziert und erhält die neuen Namen „Test1“, „Test2“, … , „Test9“. Der Zähler, also der Wert, der in dieser Schleife läuft, wird zu Beginn mit `i=1` initialisiert. Dann folgt die Bedingung `i<10`. Solange diese Bedingung erfüllt ist, wird die Schleife immer wieder erneut abgearbeitet. Da in diesem Fall die Schleife aber unendlich lange

laufen würde, muss mittels `i++` der Wert für i verändert werden. `i++` erhöht i bei jedem Aufruf um eins. Dies erklärt auch, warum der Aufruf `duplicateMovieClip` innerhalb der Schleife neunmal aufgerufen wurde.

Eine weitere Möglichkeit ist der Aufruf der so genannten **while-Schleifen**. Diese sind eine verstärkte Mischung aus if-Abfrage und for-Schleife. Alle Aufrufe innerhalb der while-Schleife werden so lange abgearbeitet solange eine beliebige Bedingung erfüllt ist. Es erfolgt eine laufende Abfrage der Bedingung, auf die die while-Schleife im Gegensatz zur for-Schleife keinen Einfluss nimmt. Folgendes Beispiel veranschaulicht Ihnen dies:

```
k=0;
while(k!=10){
k++;
}
```

Diese Schleife wird zehnmal abgearbeitet, da *k* mit 0 initialisiert wurde und die Bedingung `k!=10` (k ungleich 10) lautet. *k* wurde bei jedem Durchlauf um eins erhöht.

2 Neuerungen in Flash 5

Hier erhält der erfahrene Flash-4-User einen kurzen Überblick über die neuen Funktionen von Flash 5.

Neben einer deutlich verbesserten Oberfläche verfügt Flash 5 über eine Menge neuer Funktionen, seien es `Mouse.hide()` (Mauszeiger verbergen), trigonometrische Funktionen wie `Math.sin(wert)` (den Sinuswert eines Wertes berechnen), die Möglichkeit des Druckens aus dem Flashplayer oder gar das XML-connect. Es hat sich auf jeden Fall viel getan – nicht zuletzt wurden viele Bugs bereinigt.

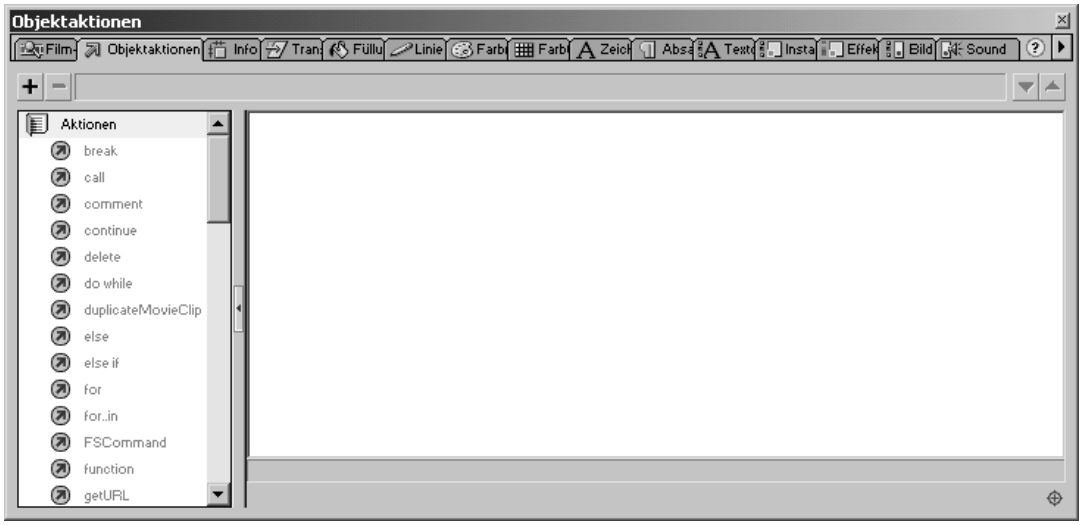
Jeder der von Flash 4 auf Flash 5 umsteigt, wird sofort nach dem Starten von Flash 5 feststellen, dass alles irgendwie anders ist.

2.1 Die Bedienoberfläche

Die Oberfläche wurde komplett überarbeitet. Jede Option hat jetzt ihr eigenes Fenster (Bedienfeld). Erfreulich für jeden, der mindestens einen 19-Zoll-Monitor besitzt, weniger erfreulich für den Rest, da die Bedienfelder einen großen Teil der Bildschirmfläche beanspruchen. Flash 5 bietet aber die Möglichkeit, die Oberfläche individuell anzupassen. Hierbei können die Fenster (Bedienfelder) per Drag and Drop beliebig kombiniert werden. Wer Flash 4 kennt, ist es gewohnt, die meisten Befehle über die rechte Maustaste aufzurufen oder diese neben dem Aktionscriptfenster zu suchen. Deshalb gibt es auf der CD einen Bedienfeldsatz Namens *allInOne,* der genau diese Voreinstellungen besitzt. Bei diesem Bedienfeldsatz finden Sie alle Optionen in einem Fenster wieder. Flash-4-User sind es gewohnt, immer das Aktionscriptfenster für alle möglichen Operationen zu öffnen, so dass diese sich hier wahrscheinlich sofort wieder zu Hause fühlen.

Ordner *allInOne*

Um die Panelsettings (Bedienfeldsätze) nutzen zu können, müssen Sie die Dateien von der CD in Ihr Flash-5-Panelsettingsverzeichnis kopieren. Danach kann man, bei geöffneten Projekt, unter FENSTER › BEDIENFELDSÄTZE die neuen Bedienfeldsätze aktivieren. Eigene Bedienfeldsätze können Sie unter diesem Optionsmenü auch abspeichern oder aufrufen.

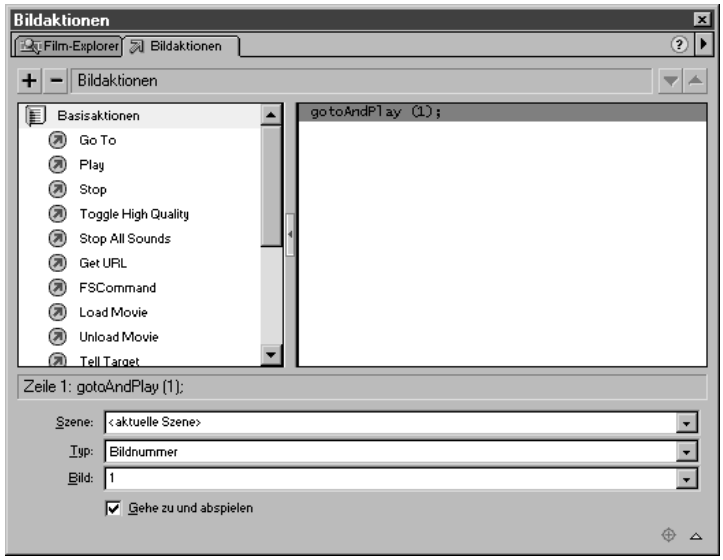


2.2 Die Zeitleiste

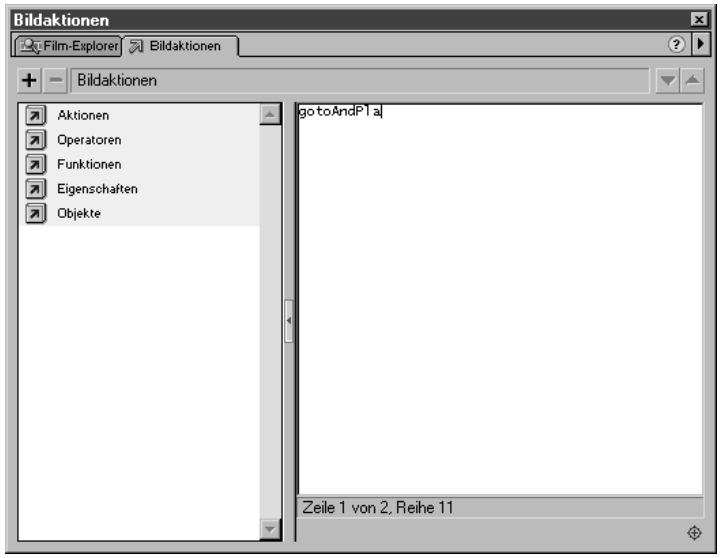
Die Bedienung der Zeitleiste hat sich ebenfalls verändert. Sie können sie jedoch unter dem Menüpunkt BEARBEITEN › EINSTELLUNGEN › ALLGEMEIN › ZEITLEISTENOPTIONEN wieder auf den Stil von Flash 4 zurückstellen.

2.3 Expertenmodus vs. Normalmodus

Ein weiteres neues Feature ist der Expertenmodus. Hier können Sie die Entwicklungsumgebung Ihren Bedürfnissen anpassen. Kennen Sie sich bereits in Flash aus und beherrschen auch die ActionScript-Syntax einigermaßen, so können Sie per Expertenmodus auch direkt die Befehle eingeben. Dies spart viel Zeit, wenn man berücksichtigt, welcher Sprachumfang jetzt zur Verfügung steht. Dennoch hält Macromedia weiterhin am Normalmodus fest, um Einsteigern und Umsteigern die Arbeit zu erleichtern.



◀ *Normalmodus*



◀ *Expertenmodus*

2.4 Neuerungen in ActionScript

Neben der Benutzeroberfläche hat sich bei Flash 5 auch das ActionScript geändert. Sehr viele neue Befehle sind hinzugekommen, wobei die meisten unter Flash 4 mit Tricks erreicht werden konnten. Die komplette Syntax wurde der von JavaScript angeglichen. Es wurde dabei allerdings Wert darauf gelegt, dass die alten Flash-4-Befehle noch funktionieren. Allerdings ist es ratsam, auf die neue Flash-5-Syntax umzusteigen, die auf den in der ECMA-262 enthaltenen Spezifikationen basiert. Dies ist ein Dokument, das von der Vereinigung europäischer Computerhersteller (ECMA) herausgegeben wurde und als internationaler Standard für JavaScript gilt.

Das ActionScriptfenster bietet neben dem normalen Editionsmodus (anwählbar über Strg + N), den man von Flash 4 her kennt, nun auch einen Expertenmodus (anwählbar über Strg + E) mit dem direkt im Script Veränderungen vorgenommen werden können. (Diese Tastenkürzel funktionieren nur im ActionScriptfenster.) Des Weiteren ist es auch möglich, das ActionScript einzeln einzuladen oder zu exportieren, so dass es nicht mehr in die Zwischenablage kopiert und anschließend mit Hilfe eines Texteditors abgespeichert werden muss.

Im folgenden fassen wir die wichtigsten ActionScript-Neuerungen zusammen.

- **Funktionsaufrufe**

Während man bei Flash 4 noch ein eigenes Movie mit leeren Frames erstellen musste und die Frames mit `call` aufrief, um eine Art Funktionsaufruf zu erstellen, ist dies unter Flash 5 direkt möglich.

- **Objektsteuerung**

In Flash 5 ist es möglich, Objekte einmal zu definieren, um sie später wieder zu verwenden, so oft man möchte. Den gleichen Effekt konnte man unter Flash 4 auch erzeugen, indem man ein Movie samt Befehlen dupliziert hat.

Colorobjekt
→ *6.1 Memory.*

- **Colorobjekt**

Mit dem Colorobjekt hat man die Möglichkeit, die Farbe einer Grafik zu ändern.

Soundobjekt
→ *74 Soundeffekte.*

- **Soundobjekt**

Mit dem Soundobjekt hat man Einfluss auf das Abspielverhalten eines bestimmten Sounds: Man kann Lautstärke, Balance und Ausgabekanäle per ActionScript verändern.

Stringobjekt
→ *82 Externe Dateien einlesen – Zeichen umwandeln.*

- **Stringobjekt**

In Flash 4 konnte man zwar mit einem String auch Operationen durchführen. Durch das Stringobjekt gibt es aber nun neue Befehle, die den Umgang mit Strings erleichtern: Beispielsweise kann man bestimmte Zeichen aus Stringketten auslesen oder löschen, was in Flash 4 nur über Umwege möglich war.

- **Dateobjekt**

Bei Flash 4 musste man die aktuelle Uhrzeit importieren, was meist über JavaScript geschah. Bei Flash 5 gibt es jetzt ein Dateobjekt, das eine ganze Fülle an Werten liefert – von der lokalen Uhrzeit und dem Datum bis hin zur Weltzeit.

Dateobjekt
→ *6.1 Memory – Anzeige der Uhrzeit.*

- **Matheobjekt**

Während es in Flash 4 noch sehr schwierig war, Winkelfunktionen oder andere mathematische Funktionen anzuwenden, besitzt Flash 5 dafür ein Matheobjekt, so dass man nicht mehr gezwungen ist, den Sinus über Umwege erst selbst zu errechnen.

Matheobjekt
→ *Kapitel 1: Einführung.*

- **Selections**

Durch diesen neuen Befehl ist man nun in der Lage, die Reihenfolge der Auswahl der Textboxen durch die Tabulatortaste direkt zu verändern. Dies war unter Flash 4 nicht möglich.

- **MouseHide**

Während man in Flash 4 vergebens versucht hat, mit JavaScript den Mauszeiger zu verstecken, bietet Flash 5 nun einen eigenen Befehl dafür.

MouseHide
→ *70 Mauseffekte/ Maustrailer.*

- **XML**

Flash 5 bietet eine Anbindung von XML-Dateien. Diese kann man unter Flash generieren oder aus Dateien auslesen.

XML
→ *89 XML.*

- **HTML**

In Flash 5 ist es nun möglich, HTML-Text einzuladen – allerdings mit Abstrichen, da dieser Text doch gewissen Spezifikationen unterliegen muss: Zum einen muss am Anfang des Textes eine Variablenzuweisung stehen, und zwar *Var1=Ihr HTML-Text*. Zum anderen können nicht alle HTML-Befehle eingelesen werden. Der eingeladene HTML-Text wird in Flash dann in einem Textfeld abgebildet.

HTML
→ *82 Externe Dateien einlesen – HTML.*

- **Swap depth**

Mit dem Z-Index war es unter Flash 4 möglich, die Tiefen von Movies zu tauschen: In einem „Array“ mit dem Namen `z_index` waren die Tiefen der einzelnen Movieclips gespeichert. Wenn man diese nun ändern wollte, konnte man alle Movies noch einmal duplizieren und diesen die neuen Tiefen zuweisen. Bei Flash 5 gibt es dafür nun die Funktion `.swapDepths(target)`, mit der die Tiefen von zwei Movieclips vertauscht werden können.

- **HitTest**

In Flash 5 gibt es jetzt einen Befehl, mit dem direkt überprüft werden kann, ob sich zwei Movies berühren bzw. überlappen. In Flash 4 musste man für eine Kolisionsdetection noch mehrere komplizierte If-Abfragen erstellen.

Hittest
→ *6.4 Einbinden von zusätzlichen Movies (Objekten) – Kolisionsdetection.*

- **attachMovie**

Neben dem DuplicateMovie-Befehl gibt es in Flash 5 nun auch einen `attachMovie`-Befehl. Dieser dupliziert einen Movieclip und hängt ihm einen anderen Movieclip an. Zudem muss das anzuhängende Movie nicht auf der Bühne liegen, sondern kann sich in der Bibliothek (Library) befinden, da der `attachMovie`-Befehl mit Verknüpfungsnamen arbeitet.

- **getBounds**

Diese Funktion stellt eine große Arbeitserleichterung dar: Wo man früher noch über die Breite und X-Position den Rand eines Movieclips errechnen musste, erhält man nun mit der Funktion `getBounds` direkt `xMax`, `xMin`, `yMay` und `yMin`.

GetBytesLoaded...
→ *46 Preloader – GetBytesLoaded().*

- **getBytesTotal und getBytesLoaded**

Mit diesen zwei Funktionen kann man die Gesamtgröße der SWF-Datei beziehungsweise die Anzahl der bereits eingeladenen Bytes ermitteln. In Flash 4 war dies nur mit Hilfe von JavaScript möglich.

- **onClipEvent**

Dieser neue Befehl ermöglicht es, einem Movie direkt ein ActionScript zuzuordnen. Das Script wird nicht in die Frameleiste gestellt, sondern direkt an das Movie angehängt. Man lässt hierzu das ActionScriptfenster offen und wählt ein Movie aus. Die entsprechenden Befehle sind unabhängig vom Abspiel-Status des Movies. Sie werden je nach der Art des Events zu verschiedenen Zeitpunkten ausgeführt. Dadurch ist es nicht notwendig, mit Frameloops, komplizierten TellTarget-Aktionen mit Goto-Befehlen oder call-Aktionen zu arbeiten. Die Befehle müssen in einem *Handler* stehen – nämlich in einem `onClipEvent`. Es gibt neun verschiedene Argumente für das `onClipEvent`:

- load: Die zugehörige Aktion wird einmal direkt beim Start des Movies ausgeführt, dies ist vergleichbar mit einem ActionScript, das direkt im ersten Frame eines Movies steht, wobei das erste Frame nur einmal aufgerufen wird.
- unload: Die zugehörige Aktion wird beim Entladen des entsprechenden Movies ausgeführt.
- enterFrame: Der Effekt ist vergleichbar mit dem eines Frameloops: Die entsprechende Aktion wird ausgeführt, sobald ein neuer Frame abgespielt wird. Dies hat jedoch nichts mit dem Abspielverhalten des Movies zu tun – ob das Movie z. B. durch `stop()` angehalten wurde, ist für `enterFrame` nicht relevant.
- mouseMove: Die zugehörige Aktion wird ausgeführt, sobald die Maus bewegt wird.
- mouseDown: Die zugehörige Aktion wird ausgeführt, sobald die Maustaste gedrückt wird – dies ist vergleichbar mit `on(press)` bei einem Button, nur dass hier eben kein Button benötigt wird.
- mouseUp : Die zugehörige Aktion wird ausgeführt, sobald die Maustaste wieder losgelassen wird – vergleichbar mit `on(release)` bei einem Button.
- KeyDown: Die zugehörige Aktion wird beim Drücken einer Taste ausgeführt. Der Einsatz ist vor allem bei einmaligen Aktionen sinnvoll. Die Abfrage wird auch ausgeführt, wenn eine Taste länger gedrückt wird. Die späteren Aktionen werden schneller ausgeführt als die erste Aktion, was nach

dem ersten Tastendruck einer kleineren Verzögerung entspricht. Für eine kontinuierliche Abfrage verwenden Sie bitte das Key-Objekt.

- KeyUp: Die zugehörige Aktion wird beim Loslassen einer Taste ausgeführt.
- Data: Die zugehörige Aktion wird ausgeführt, wenn mit `loadVariables()` oder `loadMovie()` Variablen oder Movies eingeladen wurden. Bei `loadVariables()` wird sie einmal nach Beendigung des Einladevorgangs aufgerufen. Bei `loadMovie()` wird die Aktion mehrmals ausgeführt, da ein Movie in mehrere Abschnitte aufgeteilt wird.

- **Keyboardabfrage**

Jeder, der versucht hat, ein Spiel unter Flash 4 zu erstellen, wird sich über die beschränkten Möglichkeiten der Tastaturabfrage geärgert haben. In Flash 5 gibt es ein neues System, um abzufragen, ob eine Taste gedrückt ist oder wieder losgelassen wurde.

Hittest
→ *6.4 Einbinden von zusätzlichen Movies (Objekten) – Abfrage der Tastatur.*

- **Punktsyntax**

In Flash 4 hat man auf die darüber liegenden Movies mit `../movie/:variable` verwiesen, in Flash 5 kann man statt der zwei Punkte auch `_parent` benutzen, also `_parten.movie.variable`.

Punkt-Syntax
→ *Kapitel 1: Einführung.*

- Durch die Punkt-Sytax ist auch der Befehl `getProperty` überflüssig geworden, da man direkt auf die Eigenschaften zugreifen kann – so etwa auf die y-Position des Movies: `root.Movie._y`
- **Änderungen, Austausch der Befehle** `_level0` kann nun auch `_root` heißen.

  Statt `and` kann man auch `&&` schreiben. Für `or` funktioniert auch `||`.

  `&` zum Verketten von Variablen und Strings wurde durch `add` ersetzt. Statt `add` ist es aber auch möglich, ein Pluszeichen (+) zu benutzen.

  Die If-Bedienungen wurden JavaScript angeglichen und müssen deshalb mit == geschrieben werden. Flash gibt keine Fehlermeldung aus, wenn man wie in Flash 4 nur ein Gleichheitszeichen benutzt, aber das ActionScript funktioniert dennoch nicht.

  Kleinere IF-Abfragen können in Flash 5 nun auch mit `?:` gestaltet werden.

```
If( Wert > 5){
x=1;
} else {
x=2;
}
```

Dies könnte man in Flash 5 nun auch folgendermaßen realisieren:

```
x = Wert>5 ? 1 : 2;
```

In allgemeinen Begriffen ausgedrückt:

Wert = Bedingung? Zuweisungs-variable/wert bei True : Zuweisungs-variable/wert bei false;

Man kann dies auch mit *= oder += usw. kombinieren. Bei größeren If-Schleifen empfiehlt sich aber wieder die normale, übersichtlichere Syntax.

In Flash 5 gibt es nun auch eine andere Schreibweise für `eval()`: Angenommen in dem Film existieren mehrere Movies, *Movie1 bis Movie9*. Wenn man diese Movies aus einer for-Schleife ansprechen möchte, geht dies normalerweise über:

```
eval("_root.movie"+number+".text") = "text";
```

wobei man voraussetzt, das `number` bis 9 hochgezählt wird.

In Flash 5 kann man dies allerdings auch so schreiben:

```
_root["movie"+number].text = "text";
```

Was vielleicht verwirrend wirkt, ist dass vor der eckigen Klammer kein Punkt steht. Wenn man sich einmal dieses Beispiel anschaut, wird klar, dass bei der Schreibweise mit den eckigen Klammern immer eine Pfadangabe voran stehen muss.

```
z = 333;
y = "z";
x = eval(y);
```

Mit Klammerschreibweise:

```
z = 333;
y = "z";
x = this[y];
```

Dies sind eine ganze Menge nützlicher Neuerungen. Nach einer kurzen Einarbeitungszeit fühlt man sich schnell wieder wie zu Hause und der Umstieg hat sich gelohnt.

2.5 Kompatibilität Flash 4/Flash 5

Flash-4-Files mit der Programmversion 5 erstellen

Mit dem Flash-5-Player existiert wieder ein neuer Standard für Shockwave-Files. Obwohl die Installation des Flash-5-Plugins schnell und einfach vonstatten geht, hat es sich noch nicht überall durchgesetzt. Insofern kann es in dem einen oder anderen Fall sinnvoll sein, Flash-4-Files zu erstellen, die vom Flash-4-Player interpretiert werden können und so eine größere Zahl von Nutzern erreichen.

Mit Flash 5 kann man problemlos Flash-4-Files erstellen. Sie können dabei allerdings nicht auf alle neuen Befehle von Flash 5 zurückgreifen, da der Flash-4-Player diese zum Teil nicht unterstützt.

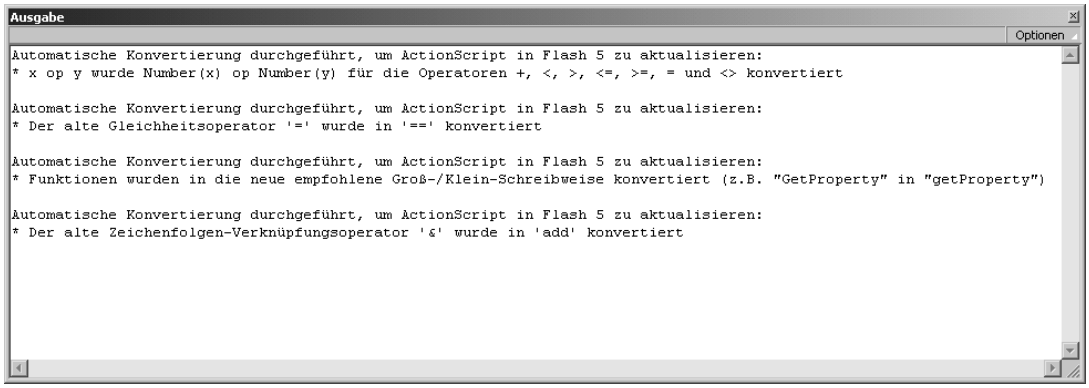
Flash 5 bietet hierbei gegenüber der Vorgängerversion neben der überarbeiteten Oberfläche Vorteile beim Zeichnen von Bezierkurven und beim Rechnen mit mathematischen Funktionen (Kosinus , Sinus).

Flash-4-Files mit Flash 5 öffnen

Flash-5-Befehle in Flash 4 nutzen
→ *Kapitel 8: Flash Workarounds.*

Problematischer ist es hingegen, wenn Sie Flash-4-Files in Flash 5 aufrufen möchten. Flash 5 konvertiert die Dateien zwar, jedoch nicht immer problemlos: Das dabei erzeugte ActionScript ist unter Umständen nicht mehr sauber aufgebaut. Da es gerade bei umfangreicheren ActionScripts auf einen klar gegliederten Aufbau ankommt, empfiehlt es sich oft, die Datei neu zu erstellen.

Wer einmal versucht hat, ein Flash-4-File unter Flash 5 zu öffnen, wird das folgende Bild gut kennen:



Flash 5 konvertiert dabei folgende Befehle:

1. Mathematischen Vergleichsoperationen mit Variablen wird ein `Number()` voran gestellt. `x op y` wurde `Number(x) op Number(y)`
2. Vergleichsoperationen werden umgewandelt. = zu ==
3. Die Groß- und Kleinschreibung von Befehlen wird der neuen Syntax angepasst, z. B. `GetProperty` in `getProperty`
4. Der alte Zeichenfolgen-Verknüpfungsoperand & wird in `add` konvertiert.
5. Flash 5 benötigt teilweise zusätzliche Klammern für Funktionsaufrufe. Diese werden hinzugefügt.

Erstellen von Flash-5-Files
→ *8.7 Tipps / Wissenwertes.*

Die meisten Fehler entstehen dadurch, dass Sonderzeichen in Variablennamen verwendet werden oder Funktionen bzw. Objekte mit den gleichen Bezeichnungen benannt sind wie Variablen. Man sollte stets darauf achten, dies zu vermeiden. Insofern ist es auch sinnvoll, längere Bezeichnungen für Variablen zu benutzen. Flash gibt im Fall einer Doppelkennzeichnung auch keine eindeutige Fehlermeldung aus, so dass nicht sofort zu erkennen ist, warum ein Programm nicht läuft. – Im übrigen sollte man auch darauf achten, keine Flash-5-Schlüsselwörter (Befehle) für die Benennung von Variablen zu benutzen.

Noch erwähnenswert ist, dass mit Flash 5 auch eine neue Version des Shockwaveplayers herausgebracht wurde. Da die Anzahl der Frames, die in einer Sekunde maximal abgespielt werden können, von der Leistungsfähigkeit des Rechners abhängt, hat der ältere Player maximal 14–16 Fps (Fra-

mes pro Sekunde) unterstützt. Der neue Player ist darauf angelegt, auch höhere Framezahlen zu unterstützen. Dies kann allerdings dazu führen kann, dass ältere SWFs, die auf eine hohe, mit dem alten Player gar nicht erreichbare Abspielrate eingestellt sind, mit dem neuen Player zu schnell abgespielt werden.

2.6 Drucken aus Flash

Flash 5 bietet nun endlich einen direkten Druckbefehl:

```
print(Instanzname, "bmovie");
print(Instanzname, "bmax");
print(Instanzname, "bframe");
```

Im ersten Fall wird mit der Option `bmovie` die Instanz entsprechend ihrer tatsächlichen Größe auf der Hauptbühne gedruckt. Mit der Option `bmax` wird der Inhalt der Instanz auf das Seitenmaximum proportional skaliert, und mit der Option `bframe` wird der Rahmen der Instanz als maximale Breite bzw. Höhe genommen. Per `print` werden die Daten vektorbasiert an den Drucker geschickt. Bei Verwendung von `printAsBitmap` werden die Bilddaten gerastert, also als Bitmap an den Drucker übermittelt.

3 Begriffserklärungen

Um Missverständnissen und Unklarheiten vorzubeugen, werden wir im Folgenden einige grundlegende Begriffe definieren, damit eindeutig ist, was damit jeweils gemeint ist.

▶ Aktionen

Der Begriff bezeichnet Aktivitäten oder Interaktionen eines Objektes mit ggf. anderen Objekten. Das hiermit verbundene ActionScript wird mit einem kleinen „a“ über dem dazugehörigen Frame gekennzeichnet.

▶ Bibliothek

→ *Library.*

▶ Boolean

Boolean ist die Bezeichnung für die Werte `true` oder `false`. Wenn eine Funktion als Rückgabewert `boolean` liefert oder fordert, sind `true` bzw. `false` gültige Eingabe- bzw. Rückgabewerte.

▶ Bühne

Die Bühne ist der aktive Bereich des Flashfilms, unabhängig davon, wo man sich im Film befindet.

▶ Button

Zu deutsch „Knopf“, hat in Verbindung mit Flash, aber auch mit HTML, DHTML, JavaScript usw. verschiedene Eigenschaften:

- Das „normale“ Aussehen (o),
- eine Aktion oder ein Aussehen für den Event `mouseOver` (z. B. Button wird blau)
- eine Aktion oder ein Aussehen für den Event `mouseDown` (z. B. gehe zu URL: http://ww...)
- eine Aktion oder ein Aussehen für den Event `mouseOut` (zumeist nur DHTML, bzw. JavaScript: stelle altes Aussehen wieder her – macht Flash automatisch)
- und einen aktiven Bereich.

▶ Deklarieren

Wird eine Variable deklariert, so wird ihr ein Wert zugeordnet. Dieser Wert kann `boolean`, `integer`, `string` oder `real` sein.

▶ Dekrement

Dieses bezeichnet eine neue Funktion in Flash die aus JavaScript übernommen wurde. Sie verkleinert einen Integer-Wert bei jedem Aufruf um 1. Wenn i = 10, dann ist i-- = 9. Diesen Zähler nennt man Dekrement.

▶ Ebene

Mit Ebene sind immer die Ebenen angesprochen, auf denen direkt gearbeitet wird, die auch benannt werden und deren Reihenfolge per Drag and Drop geändert werden kann. Sie sind an dieser Stelle klar von Layern zu unterscheiden, da diese in einem anderen Kontext verwendet werden.

▶ Event

Bezeichnet zumeist ein Ereignis, für das eine bestimmte Aktion eintreten soll.

▶ Film

Film bezeichnet die gesamte Flash-Datei.

▶ **Frame**
Als Frame wird ein einzelnes Bildsegment aus der Zeitleiste bezeichnet – quasi die kleinste Zeiteinheit in Flash.

▶ **Frameloop**
Als Frameloop werden mehrer Frames bezeichnet, die nacheinander ablaufen und bei dem im letzten Frame wieder eine Anweisung steht, die den Abspielkopf wieder in das erste Frame des Frameloops zum abspielen setzt.

▶ **Grafik**
Hier gilt es zwischen Vektorgrafik und Pixelgrafik zu unterscheiden. Vektorgrafiken sind wie der Name sagt durch Vektoren definiert (→ *Vektor/Pfad*) und weisen im Gegensatz zu Pixelgrafiken (.jpg, .png, .gif, ...), die sich aus einer Menge einzelner Bildpunkte zusammensetzen, bei der Skalierung keine Qualitätsverluste auf.

▶ **Hauptbühne**
Mit Hauptbühne ist der aktive Bereich des Flashfilms bezeichnet, der sich in der Hierarchie über allen anderen befindet. Die Hautbühne samt Darsteller (→ *Objekte*) wird beim Aufrufen des Films sichtbar.

▶ **Inkrement**
Dieses bezeichnet eine neue Funktion in Flash, die aus JavaScript übernommen wurde. Sie erhöht einen Integer-Wert bei jedem Aufruf um 1. Wenn i = 10, dann ist i++ = 11. Diesen Zähler nennt man Inkrement.

▶ **Instanz**
Eine Instanz ist ein Objekt in der Library (Bibliothek). Man kann jedes Objekt dazu machen (markieren + F8), oder direkt ein neues anlegen (Strg + F8). Legt man eine Neue an muss man sich zwischen Grafik, Button und Film entscheiden.

▶ **Integer**
Wert ohne Nachkommastellen.

▶ **Keyframe**
Ein Keyframe beinhaltet im Gegensatz zum Frame meist neue oder neu positionierte Objekte, oder einfach nur eine ActionScript-Anweisung.

▶ **Label**
Mit Label wird der Name eines Schlüsselbildes, oder Keyframes bezeichnet. Auf der Arbeitsoberfläche sind Frames, die mit einem Label (Namen) bezeichnet sind, mit einem kleinen Fähnchen gekennzeichnet.

▶ **Level**
Level bezeichnen die übergeordneten Ebenenstrukturen, die im Zusammenhang mit `loadMovie` verwendet werden. Jeder Level kann beliebig viele Ebenen enthalten.

▶ **Library**
Kurzschreibweise für Bibliothek (Strg + L). Dort befinden sich alle verwendeten Instanzen, Grafiken und Sounds, ob aktiv oder inaktiv.

▶ **Maskenebene**
Eine Maskenebene ist eine Art Filter für untergeordnete Ebenen. Alle Objekte auf dieser Ebene verschwinden bei Aktivierung der Maske, an dieser Stelle werden aber die Objekte der untergeordneten Ebenen wieder sichtbar. Mit einem Kreis auf der Maskenebene erzeugen Sie z. B. einen Gucklocheffekt.

▶ **Objekt**
Mit Objekt ist an dieser Stelle kein Objekt im programmiertechnischen Sinn gemeint, sondern eine Auswahl von Elementen, nicht zwangsläufig Instanzen.

▶ **Pfad**
Pfade sind der Kern eines jeden vektororientierten Objekts. Sie definieren eine Kurve nur anhand von Richtungsänderungen (Vektoren).

▶ **Pfadebene**
Die Pfadebene hat Einfluss auf Tweenings sofern erwünscht. An dem auf der Pfadebene enthaltenem Pfad kann man die Bewegung eines Objektes ausrichten. Auf diese Weise erspart man sich den Umweg über viele Keyframes.

▶ **Real**
Bezeichnet einen Fließkommawert.

▶ **Schlüsselbild**
Ein Schlüsselbild bezeichnet einen Frame, in dem alle für ein Tweening maßgeblichen Objektpositionen und Eigenschaften festgelegt sind.

▶ **Sound**
Bezeichnung für einen Klang, Ton oder Musik.

▶ **Source**
Hinter diesem Begriff verbirgt sich der Quellcode des ActionScripts. Da wir im Verlauf des Buches auch mit JavaScript arbeiten, möchten wir an dieser Stelle für die Flash-eigene Sprache nicht den Ausdruck Script verwenden.

▶ **String**
Bezeichnet eine Zeichenkette.

▶ **Szene**
Eine Szene ist ein Teil des Filmes. Schaltet man z.B. einer Flash-Site einen Preloader vor, empfiehlt es sich, diesen in eine Szene zu setzen, die sich vor den eigentlichen Inhaltsszenen befindet. Alle Szenen werden in einem Film ausgegeben.

▶ **Tag**
Ein Tag ist ein abgeschlossener HTML-Befehl, man spricht auch von geöffneten (Bsp. `<a href="#">test`) und geschlossenen Tags (Bsp. `<a href="#">test</a>`).

▶ **Target**
Das Target bezeichnet das Ziel, in das Objekte, auf die verwiesen wurde, geladen werden. Dies kann eine Seite in HTML sein, die in einem Top-Frame geladen wird, oder ein Flash-Film, der in eine andere Instanz geladen wird.

▶ **Tween**
Es bezeichnet die kleinste ActionScript-unabhängige Animationseinheit. Es gibt Form- und Bewegungstweenings, die sich durch das Verformen bzw. das Positionieren der Objekte unterscheiden. Bewegungstweens sind durch eine violette Farbhinterlegung, Formtweens durch eine grüne Farbhinterlegung der betroffenen Felder gekennzeichnet.

▶ **Variable**
Eine Variable ist ein Platzhalter für einen ihr zugeordneten Wert. Zugeordnet werden können boolean, integer, string oder real.

▶ **Vektor**
Ein Vektor soll hier nur für den zweidimensionalen Raum als Richtungsänderung und nicht aus mathematischer Sicht betrachtet werden.

Hinweis:
Begriffe wie INSTANZNAME oder EVENT fungieren im Folgenden als Platzhalter für eine konkrete Instanz bzw. ein konkretes Ereignis. Auch „server.de“, „Name.txt“ oder „E-Mail-Adresse“ stehen jeweils für eine bestimmte URL, einen Dateinamen oder eine E-Mail-Adresse.

Pearson Education

ADDISON-WESLEY

4 Flash im täglichen Einsatz – Projektplanung

Im Folgenden möchten wir kurz auf die Planung von Flash-Projekten eingehen. Wir erstellen eine Checkliste, die Ihnen den täglichen Umgang mit Flash erleichtert. Zunächst muss berücksichtigt werden, dass es sich bei Flash-Dateien immer um gestreamte Daten handelt. Beinhaltet ein Flash-Film Elemente mit großen Datenmengen – wie z. B. Bildinformationen –, ist Vorsicht geboten, da diese zunächst einmal mit übertragen werden. Wenn nicht durch Aufteilung des Filmes oder geschicktes Preloading die Ladelast des Clients gemanagt wird, kann dies zu sehr langen Ladezeiten führen, was auf die Besucher von Websites in der Regel sehr abschreckend wirkt. Hier folgt eine Liste von „Do"s und „Don't"s, die nicht den Anspruch auf Vollständigkeit erhebt. Die Liste ist nicht Flash-spezifisch, gehört aber zu jeder guten Siteplanung.

:-) To Do

▶ **Do:**

- *Schlüssige Benutzerführung*: Führen Sie den Benutzer so durch die Site, dass themenverwandte Links immer zugänglich sind, z. B. ein Hyperlink von der Warenkorbseite zu den Allgemeinen Geschäftsbedingungen.
- *CI* – Corporate Identity: Achten Sie darauf, dass das Layout einheitliche Elemente enthält – wie z. B. ein Firmenlogo. Corporate Identity heißt hier in erster Linie Verwendung der firmeneigenen Farben, Fonts und Logos.
- *Schriftartenauswahl*: Achten Sie auf den sparsamen Einsatz unterschiedlicher Fonts. Sowohl Schriftart als auch Schriftgröße sollten nicht zu sehr variieren. Lesbarkeit und Übersichtlichkeit sind hier oberstes Ziel. Generell sind serifenlose Schriftarten auf dem Bildschirm besser lesbar. Je nach Thema bzw. Aussage können aber auch andere Schriftarten sinnvoll sein.
- *Eindeutigkeit*: Machen Sie dem Benutzer die Unterscheidung von Navigation und Fließtext so einfach wie möglich.
- *Transparenz*: Verdeutlichen Sie bereits auf der Introseite, wer Sie sind und was Sie machen. Wenn Sie eine Seite mit Flash entwickeln, denken Sie auch an Benutzer mit alten Computern oder niedrigen Übertragungsraten. Zumindest Anschrift und Telefonnummer sollten auf Anhieb zu finden sein.

- *Innovation*: Eine witzige Idee, einen Benutzer durch die Sitestruktur zu führen, ist immer gut. Solange der Weg zum Gesuchten nicht über zu viele Mausklicks führt oder die Navigation labyrinthartig ausartet, ist alles erlaubt.
- *Kurze Wege*: Führen Sie Ihre Besucher geschickt, schnell und präzise durch die Site. Achten Sie darauf, dass die Wege zu *jeder* Seite Ihrer Webpräsenz nicht zu lange werden.

▶ **Don't:**

- *Sackgassen*: Führen Sie den Benutzer nie in so genannte Sackgassen, aus denen er nur durch ein „Zurück" entkommt.
- *Reizüberflutung* – Achten Sie darauf, dass das Augenmerk des Sitebesuchers nicht zu sehr vom Wesentlichen abgelenkt wird. Wenn Sie einen Text blinken und den anderen hereinfliegen lassen, wird der Besucher der inhaltlichen Aussage unter Umständen in beiden Fällen keine erhöhte Aufmerksamkeit widmen.
- *Überforderung*: Beschränken Sie sich auf eine begrenzte Zahl von Auswahlmöglichkeiten. Wer in einer Produktauswahl zwischen 500 verschiedenen Elementen wählen kann, wird voraussichtlich keine Entscheidung treffen. Führen Sie Ihre Besucher lieber durch Ihr Angebot.

Planen Sie sorgfältig bereits im Vorfeld und bewerten Sie die oben angeführten Punkte für jede Site individuell. Dies erspart im Nachhinein eine Menge Aufwand. Legen Sie alle Daten so an, dass keine langen Ladezeiten entstehen. Wenn Sie z. B. eine Galerie in Flash umsetzen oder für einen Shop Produktbilder einbinden, muss berücksichtigt werden, dass Bilddaten viel Speicher brauchen. Die dadurch verursachten langen Ladezeiten lassen sich durch das Auslagern der Bilder umgehen. Sie können die Bilder entweder in einzelne Flash-Filme legen und diese mittels *LoadMovie* einladen, oder Sie legen ein JavaScript an, dass Ihnen ein neues Fenster mit dem entsprechenden Bild öffnet. Auf diese Weise werden Bilder erst bei Bedarf übertragen. Im Kapitel *Preloading* werden Sie auch mit Techniken vertraut gemacht Bilder zu laden, während andere Bereiche der Site gerade keinen Datentransfer verursachen.

→ *Preloading*

In den folgenden Kapiteln werden Sie viele Möglichkeiten zur effektvollen Gestaltung von Flash-Sites kennen lernen, doch beachten Sie, dass manchmal weniger mehr ist!

5 Der Pool – Anwendungen

Der Pool ist eine umfassende Sammlung von Flash-Lösungen für das Internet. Der modulare Aufbau verschafft Ihnen die Möglichkeit, Zusammenhänge des ActionScripts zu erfassen und dessen Integration in andere Umgebungen umzusetzen.

5.1 Grundelemente

Mailfunktionen

Durch den Befehl `mailto` ist es möglich, den Standard-E-Mail-Client des Besuchers aufzurufen und diesem Texte zu übermitteln. Der Besucher kann den Text gegebenenfalls ändern und diesen anschließend mit seinem E-Mail-Programm verschicken.

```
GetURL(mailto:IhreEmailAdresse@Server.de);
```

Dieser Befehl ruft direkt den Standard-E-Mail-Client des Besuchers auf. Sie können diesen Befehl direkt in der Frameleiste einbauen, so dass, wenn der Besucher ein gewisses Frame erreicht hat, automatisch das E-Mail-Programm aufgerufen wird. Die Mehrzahl der Besucher wird es allerdings als unhöflich empfinden, wenn plötzlich ohne Abfrage ihr E-Mail-Client geöffnet wird. Deshalb ist es sicherlich ratsam, den Befehl in Verbindung mit einem davor geschalteten Dialogfeld zu verwenden.

```
on (release) {
GetURL(mailto:IhreEmailAdresse@Server.de)
}
```

Mit diesem Befehl wird ein leeres E-Mail-Formular mit der Empfängeradresse *IhreEmailAdresse@Server.de* geöffnet. Wenn Sie nun direkt den Betreff oder einen Inhalt vorgeben möchten, sieht dieser Befehl folgendermaßen aus:

```
on (release) {
GetURL("mailto:IhreEmailAdresse@Server.de?subject=Allgemeine
Anfrage zu den Produkten &body=Geben Sie hier bitte Ihre Anfrage
ein");
}
```

Falls Sie einen Mailto-Befehl im Flashdebugging-Modus laufen lassen, öffnet sich ein neues Browserfenster und erst danach wird das E-Mail-Programm gestartet. Dies ist ein normaler Vorgang, da der Mailto-Befehl mit get URL aufgerufen wird. Normalerweise wird die Seite schon in einem Browser angezeigt und deshalb wird kein neues Fenster geöffnet.

Die Anführungszeichen können entfallen, wenn man die Zeile nicht als Ausdruck deklariert. Diese Zeile braucht nicht als Ausdruck deklariert zu werden, da Flash keine Variablen mit an das E-Mail-Programm übergibt.

Die komplette Befehlszeile:

`mailto:IhreEmailAdresse@Server.de?subject=` Allgemeine Anfrage zu den Produkten `&body=`Geben Sie hier bitte Ihre Anfrage ein

ist keine Flash-Syntax, sondern stammt von HTML ab.

Wenn man hingegen *Betreff* und *Inhalt* mit Hilfe einer Variablen übergeben möchte, muss man den Text als Ausdruck bezeichnen und den für Flash „unwichtigen" Teil, den HTML-Befehl, in Anführungszeichen schreiben. Dadurch arbeitet Flash diesen Teil nicht ab und es kommt zu keiner Syntaxfehlermeldung.

```
on (release) {
GetURL("mailto:DeineEmailAdresse@Server.de?subject=" add
Betreff add "&body=" add Inhalt);
}
```

Stringverkettung

→ *70 Mauseffekt/ Maustrailer – Mauskoordinaten.*

Der Befehl enthält die zwei Variabelennamen `Betreff` und `Inhalt`, was man auch daran erkennt, dass diese außerhalb der Anführungszeichen stehen. Flash sieht den Text außerhalb der Anführungszeichen als „Befehle" an und den Text innerhalb als String. Mit dem Befehl `add` bzw. `+` verkettet man Strings und Variablen.

→ *43 Textfelder.*

Die zwei Variablen sollten aber auch einen Text übergeben. Entweder gibt man diesen direkt mit `SetVariable` vor, oder man gibt dem Besucher die Möglichkeit, die Variable durch ein Eingabefeld zu ändern, wie auch in der Beispieldatei auf der CD-ROM.

▶ Überprüfen der E-Mail-Adresse

Da es leider oft vorkommt, dass man sich bei Eingabe der E-Mail-Adresse vertippt, kann man mit dem folgenden kleinen ActionScript die prinzipielle Syntax der Adresse überprüfen. Geprüft wird hierbei der Aufbau, der bei jeder E-Mail-Adresse gleich ist:

xxx@xxx.xx

oder

xxx@xxx.xxx

[x steht hierbei für ein beliebiges Zeichen]

Ordner Mailfunk, Datei: mailprue.fla

Die Flash-Datei auf der CD-ROM enthält zwei Textboxen. Ein Eingabefeld für die zu prüfende E-Mail-Adresse und ein zunächst nicht sichtbares Ausgabefeld für das Ergebnis der Auswertung. Der Besucher kann im Eingabefeld eine E-Mail-Adresse eingeben und die Eingabe durch Klicken auf einen Button bestätigen, wodurch folgendes ActionScript ausgeführt wird:

```
on (release) {
    laenge = (length(eingabe));
    for (i=0; i ne laenge; i=i+1) {
        if (substring(eingabe, i, 1) eq "@") {
            AtPosition = i;
   }
   else if (substring(eingabe, i, 1) eq "." {
            PunktPosition = i;
        }
    }
    if (AtPosition>3 && (AtPosition+3)<PunktPosition && ((Punkt-
    Position+3) == laenge || (PunktPosition+2) == laenge)) {
        Ausgabe = "Eingabe gültig ";
    } else {
        Ausgabe = "Ungültige E-Mail-Adresse";
    }
}

on (release)
```

ist die Zuordnung zum Schalter, so dass das ActionScript erst durch einen Klick auf dem Button ausgelöst wird.

Buttons
→ *43 Schaltflächen/ Buttons.*

```
laenge = (length(eingabe));
```

`laenge` ist der Name der Variablen, die gesetzt wird. Mit der Eigenschaft `length()` kann man die Länge eines Strings ermitteln; in diesem Fall des Strings, der in der Variable `eingabe` steht.

Der Inhalt der Variable `eingabe` wurde durch die Eingabe der E-Mail-Adresse in das Textfeld gesetzt.

Nachdem die Länge des eingegeben Strings bzw. die Anzahl der Zeichen der E-Mail-Adresse ermittelt wurde, folgt eine for-Schleife (möglich wären auch andere Schleifen-Typen). Die Aktion in der Schleife wird für jedes Zeichen des Strings einmal wiederholt:

```
for (i=0; i < laenge; i=++)
```

Die Schleife erhöht bei jedem Durchlauf `i` um 1 und terminiert, wenn `i` nicht mehr kleiner als `laenge` ist, also wenn das Ende des Strings erreicht ist.

```
if (substring(eingabe, i, 1) eq "@")
```

Mit dem Befehl `substring` kann man auf bestimmte Teile eines Strings zugreifen. Der Befehl lässt sich folgendermaßen aufschlüsseln:
Substring (Stringname, Erster Buchstabe von links. , Länge der Kette die ausgelesen wird)

Bei diesem Beispiel wird immer nur ein Buchstabe ausgelesen, daher die Eins. Stünde dort eine Zwei, würden immer zwei Zeichen gleichzeitig verglichen. Der Vergleich des Strings fängt beim ersten Buchstaben an und

endet mit dem letzten Buchstaben, da i bis zum letzten Buchstaben hochgezählt wird und damit auch die Schleife terminiert.

Angenommen, die eingegebene E-Mail-Adresse lautet `Name@Server.de`, dann vergleicht `substring` beim ersten Durchlauf der Schleife, ob „N“ vom String `Name@server.de` gleich @ ist. Beim zweiten Durchlauf wird verglichen, ob „a“ gleich @ ist und so weiter. Beim fünften Durchlauf der Schleife wird das @-Zeichen mit dem @-Zeichen des Strings verglichen, so dass diese Abfrage ein `true` ergibt. Dadurch wird in der Variable `AtPosition = i;` die Position des @ vermerkt. Eigentlich wird dort die Anzahl der Schleifendurchläufe gespeichert, aber diese entspricht der Position des jeweils verglichenen Zeichens.

Genau das gleiche passiert mit dem Punkt (.): Die Position des letzten Punktes in dem String wird in der Variable `PunktPosition` abgespeichert. Damit endet auch die for-Schleife.

Nachdem die Dateien des Strings analysiert wurden, muss nun ihre Richtigkeit überprüft werden. Dies geschieht mit der folgenden If-Abfrage.

```
if (AtPosition>3 && (AtPosition+3)<PunktPosition && ((Punkt-
Position+3) == laenge || (PunktPosition+2) == laenge))
{
        Ausgabe = "Eingabe gültig";
    } else {
        Ausgabe = "Ungültige E-Mail-Adresse";
    }
}
```

Ausformuliert bedeutet dies: Wenn `AtPosition` größer als 2 ist und `AtPositon` plus 2 kleiner als `PunktPosition` ist und die `PunktPosition` plus 2 oder plus 3 der Länge des Strings entspricht, setze `Ausgabe` gleich „Eingabe gültig“, anderenfalls setze `Ausgabe` gleich „Ungültige E-Mail-Adresse“.

Durch denn Vergleich von

- `AtPosition>2` stellt man fest, dass sich vor dem @ mindestens drei Zeichen befinden.
- `(AtPosition+2)<PunktPosition` stellt man fest, dass sich zwischen dem @-Zeichen und dem Punkt (.) drei Zeichen befinden.
- `PunktPosition+3) == laenge || (PunktPosition+2) == laenge)` stellt man fest, dass sich hinter dem Punkt mindestens 2, aber maximal 3 Zeichen befinden.

Die Eingabe von unzulässigen Sonderzeichen hätte man auch von vornherein ausschließen können, indem man die Eingabe dieser Zeichen sperrt Textfelder

→ *43 Textfelder.*

Nur durch so ausführliche Vergleiche kann man sicherstellen, dass die eingegebene E-Mail-Adresse zulässig ist. Sicherlich hätte man diese noch genauer prüfen können, z. B. auf unzulässige Sonderzeichen oder ob wirklich nur ein @-Zeichen eingeben wurde.

Schaltflächen/Buttons

Im Folgenden möchten wir die Einbindung von Schaltflächen in Flash erläutern. Mit dem Shortcut Strg + F8 erhalten Sie ein Auswahlmenü, in dem Sie festlegen können, welche Art Instanz Sie anlegen wollen.

Wenn Sie einen Namen für die Library gewählt haben (welcher nicht automatisch den Instanznamen festlegt), werden Sie folgende Aufteilung in der Zeitleiste sehen.

Die Frames tragen nun die Bezeichnungen:

- *Auf*: Normalzustand des Buttons
- *Darüber*: neues Erscheinungsbild der Schaltfläche, wenn sich der Mauscursor darüber befindet
- *Drücken*: neues Erscheinungsbild der Schaltfläche, wenn diese gedrückt wird
- *Aktiv*: Fläche, die dem Button als aktiver Bereich zugeordnet wird. Alle Ereignisse beziehen sich auf diese Fläche.

Sie können in diese Frames auch Animationen einfügen: Bei einem Bild im Frame *Auf* beispielsweise und einer Animation im Frame *Darüber* wird der Button zu der Animation wechseln, sobald Sie mit dem Cursor über die Schaltfläche fahren.

Texteffekte
→ *68 Timeslide (Texteffekte)*

Textfelder

Will man in Flash einen Text einbinden, ist man auf die Nutzung von Textfeldern angewiesen, solange man diesen Text nicht als Grafik einfügt. Im folgenden Abschnitt erläutern wir die verschiedenen Arten von Textfeldern und ihre spezifischen Eigenschaften.

E-Mail versenden per CGI
→ *39 Mailfunktionen*

Textfelder werden für jegliche Schriftart benötigt. Wählen Sie das Textwerkzeug aus STRG + T, um auf der Bühne ein Textfeld anzulegen.

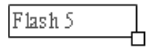
Erzeugen Sie Text möglichst mit Hilfe der Textfelder. Texteffekte können Sie auch mit Hilfe von Flash erzeugen.

Importierte Grafik mit Text sind auf jeden Fall immer größer, als ein von Flash erzeugtes Textfeld, und auch nicht so klar zu lesen.

Legen Sie ein Textfeld an, indem Sie mit gedrückter Maustaste auf die Bühne klicken und ein Feld entsprechender Breite aufziehen. Die Breite kann im nachhinein geändert werden. Wenn der Text über die maximale Breite des Textfeldes hinausgeht, wird der eingegebene Text nicht vollständig angezeigt. Mit einem einfachen Klick mit der linken Maustaste erstellt Flash ein Textfeld, das sich der Texteingabe automatisch anpasst.

Unterschieden werden diese beiden Arten von Textfeldern durch das kleine Symbol auf der rechten Seite: Ein Rechteck am oberen Rand kennzeichnet ein statisches Feld mit einer vorgegebenen Breite, ein Kreis am unteren Rand hingegen steht für ein Textfeld mit variabler Breite.

Statisches Textfeld ▶ mit fester Breite



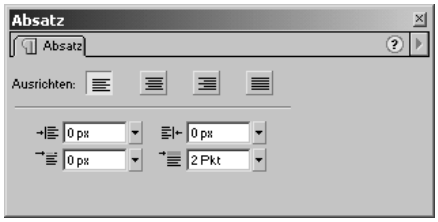
Die Unterschiede zwischen beiden Arten von Textfeldern sind auf den folgenden Seiten beschrieben.

Dynamisches Textfeld, ▶ dessen Breite sich dem eingegebenen Text anpasst.

Durch einen Doppelklick auf dieses Symbol am rechten Rand des Textfeldes können Sie von einer Variante zur anderen wechseln.

Durch ein Klicken mit der rechten Maustaste auf das Textfeld und der Auswahl des Menüpunktes BEDIENFELDER › ZEICHEN kann man die Schriftgröße, Schriftart, Schriftfarbe und den Zeichenabstand ändern. Zudem kann man dort auch die Optionen fett, kursiv, unterstrichen, hochgestellt oder tiefgestellt einstellen.

In dem Feld mit der Beschriftung „URL“ geben Sie die Ziel-URL an, mit der eine markierte Textpassage verlinkt werden soll. Mit der Option UNTERSCHNEIDEN ist gewährleistet, dass der Zeichenabstand zwischen bestimmten Buchstabenkombinationen, z. B. „A“ und „w“, verringert wird, um eine bessere optische Wirkung zu erzielen.



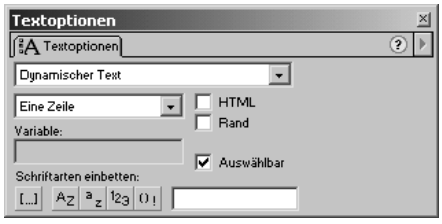
In dem Fenster ABSATZ können Sie den Text linksbündig, rechtsbündig, in Blocksatz oder zentriert ausrichten. Über die darunter liegenden Felder ist es möglich, einen Einzug vom linken bzw. rechten Rand oder einen Erstzeileneinzug einzustellen. Im Feld rechts unten können Sie den Zeilenabstand verändern.

Im Fenster TEXTOPTIONEN legen Sie fest, ob es sich um einen statischen oder dynamischen Text oder um eine Texteingabe handelt.

Statischer Text wird als Standardtext gebraucht. Wenn man einen Text eingeben möchte, der sich während des gesamten Films nicht ändern soll, wählt man diese Option. Dieses Textfeld kann man mit Tweenings bewegen und skalieren, den Inhalt aber nicht mehr ändern. Wenn Sie eine andere Schriftart wählen, die nicht zu den Flash-Geräteschriftarten gehört (_seriv, _sans, _typewriter); bettet Flash die jeweilige Schriftart ein und speichert diese mit in dem SWF. Dies ist immer sinnvoll, wenn Sie eine ungewöhnliche Schriftart wählen, da Sie nicht voraussetzen können, dass jeder Besucher diese Schrift installiert hat. Das Einbetten der Schriftart erhöht jedoch die Dateigröße. Um zu unterbinden, dass Flash Schriften einbettet, aktivieren Sie das Kästchen „Schriftart des Gerätes verwenden“ in dem Fenster TEXTOPTIONEN. Falls Sie dieses Kästchen aktiviert haben und der Besucher die ausgewählte Schriftart nicht installiert hat, wählt Flash eine auf dem Gerät des Anwenders installierte Schrift aus, die jener Schriftart am nächsten kommt. Ist jedoch eine vergleichbare Schrift nicht vorhanden, so ist nicht auszuschließen, dass dies zu einem nicht lesbaren Ergebnis führt. Die standardmäßig installierten Schriftarten sind bei kleineren Schriftgrößen unter Umständen schärfer bzw. besser lesbar und sollten deshalb bevorzugt verwendet werden.

Tweenings
→ *Kapitel 1: Einführung.*

Durch die Aktivierung des Kästchens AUSWAHL ermöglichen Sie es dem Besucher, die jeweilige Textpassage in die Zwischenablage zu kopieren und in anderen Programmen weiter zu nutzen.



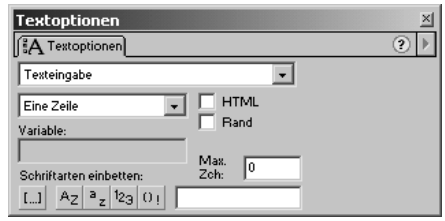
Dynamischer Text wird dazu benötigt, um Variablen auf dem Bildschirm auszugeben, die als Ausgabeboxen eingesetzt werden können. Es kann eingestellt werden, dass ein Rahmen gezeichnet wird und

ob der Text ein- oder mehrzeilig dargestellt werden soll. Legen Sie bei leeren Textfeldern immer einen Rahmen an, damit sie als solche für den Besucher auch erkennbar sind. Die Option *„Auswählbar“* hat genau die gleiche Funktion wie zuvor beim statischen Text: Sie ermöglicht es dem Besucher, Text in die Zwischenablage zu kopieren. In dem Eingabefeld „Variable“ geben Sie den Variablennamen an, der dem Textfeld zugewiesen werden soll. Wenn Sie dort *Textfeld1* eingeben und dieser Variable anschließend im Verlauf des Films den Wert 1 zuweisen, erscheint eine „1“ in diesem Textfeld. Sie können natürlich auch längere Texte eingeben wie z.B. `Textfeld1 = „Dieses hier ist das erste Textfeld";` , dabei sollten Sie darauf achten, dass der komplette Text in das Ausgabefeld passt.

HTML
→ 82 Externe Dateien einlesen.
Möchten Sie Textfelder verdrehen/verzerren, so müssen Sie die komplette Schriftart einbetten, wenn Flash sie darstellen soll.

Wenn Sie das Kästchen „HTML“ aktivieren, erwartet Flash einen HTML-Text. Diesen können Sie aus einer Variablen, die Sie zuvor schon gesetzt haben, oder aus einer externen Datei einlesen.

Mit den darunter liegenden Buttons können Sie , v.l.n.r., die komplette Schriftart einbetten oder nur Teilstücke, Grossbuchstaben, Kleinbuchstaben, Zahlen oder Sonderzeichen zulassen. Bei den vier rechten Buttons ist eine Mehrfachauswahl möglich. In der rechts daneben liegenden Eingabebox können Sie Zeichen eingeben, die das Textfeld enthalten darf. Standardmäßig kann ein Textfeld alle Zeichen enthalten. Doch sobald Sie Zeichen in das Textfeld unter Textoptionen eingeben, sind nur noch diese zulässig. Sie können z.B. ein Textfeld erzeugen, das nur die Zahlen zwischen 4–8 enthalten darf. Dazu müssten Sie in die Textbox unter Textoptionen die Zahlen 4, 5, 6, 7 und 8 eingeben (ohne Komma). Andere Zahlen würden dann nicht darstellbar sein.



Die „Texteingabe“ wird dazu benötigt, um dem Besucher die Möglichkeit zu geben, einen Text oder einen Wert an Flash zu übermitteln.
Die Optionen sind vergleichbar mit denjenigen bei statischem Text – bis auf die neue Eingabebox *„Max.Zch“:* Dort können Sie die maximale Anzahl der Zeichen eingeben, die das Textfeld enthalten kann. Überzählige Zeichen werden dann nicht abgebildet.

Preloader

Einen Preloader benötigen Sie, wenn Ihre Internetseite so groß ist, dass eine längere Ladezeit entsteht. Im Folgenden erfahren Sie, wie die Ladezeit abgefragt und dem Besucher angezeigt werden kann. Wir gehen hierbei von einer Website mit einer aufwendigen Navigation mit mehreren in Flash erstellten Untermenüs aus. Je nachdem, wie groß diese Untermenüs sind, sollte man verschiedene Preloadertechniken einsetzen. Im Folgenden zei-

gen wir, welche Technik sich in welchem Fall am besten eignet und worin die Vor- und Nachteile bestehen.

Flash-Seiten sollten so aufgebaut sein, dass sie keinen Preloader benötigen. Denn Flash wurde als *Streamingtool* konzipiert, so dass beim Öffnen der Seite sofort eine Bildschirmausgabe erfolgt und die restlichen Daten im Hintergrund eingeladen werden. Bei Seiten mit längeren Ladezeiten kann es jedoch passieren, dass der Besucher auf einen Button drückt, der auf eine noch nicht eingeladene Szene verweist. Flash springt dann entweder zu einem bereits eingeladenen Frame, was zur Folge hat, dass die gesamte Seite nicht mehr richtig reagiert, oder der User klickt und es passiert nichts. Für diesen Fall gibt es die ifFramesLoaded-Abfrage.

Die Abfrage direkt in den Button einzubauen, ist eine Möglichkeit, aber weniger nützlich, weil die Aktion nur beim Klicken einmal ausgeführt wird. Dem Besucher wird zwar gezeigt, dass die Seite geladen wird, aber nicht, wann sie komplett eingeladen ist. Als Alternative bietet es sich an, einen Preloader einzubauen. Die folgenden Abschnitte zeigen Ihnen verschiedene Möglichkeiten auf.

Bevor Sie sich für einen Preloader entscheiden, sollten Sie sich über die Größe Ihrer geplanten Website im Klaren sein.

Für kleinere Internetseiten, in die hauptsächlich Flash-Grafiken und nur wenig oder gar kein Sound integriert wurden, reicht ein einfacher Preloader aus.

Bei Seiten, die aus mehreren Untermenüs bestehen, sollten Sie abwägen, ob Sie auf einen Szenen-Preloader setzen oder die Seite in einzelne SWFs unterteilen.

Bei einem Szenen-Preloader werden automatisch sämtliche Szenen der Seite nacheinander eingeladen. Falls der Besucher sofort in das letzte Menü bzw. zur letzten Szene gelangen möchte, muss er warten, bis die anderen davor liegenden Szenen geladen sind. Wenn man die Seite hingegen in einzelne SWFs unterteilt, werden die verschiedenen Dateien bzw. Untermenüs einzeln eingeladen, und zwar erst dann, wenn man sie per Mausklick abruft. Eine weitere Variante wäre der Preloader für mehrere SWFs. Dieser kann allerdings sehr programmieraufwändig und fehleranfällig sein.

Trotz alldem sollten Sie nicht vergessen, dass der Preloader nicht der wichtigste Bestandteil einer Seite ist. Das Layout und der Inhalt sollten immer die zwei Punkte sein, in die Sie am meisten Arbeit investieren!

▶ Einfacher Preloader mit ifFramesLoaded()

Ordner preloade/standard, Datei: standard.fla

Im Prinzip unterscheidet sich der Preloader von dem obigen Beispiel mit dem Button nicht allzu sehr, aber doch in einem entscheidenden Punkt: Der Preloader wird nämlich in einem „Frameloop“ abgespielt. Das sieht folgendermaßen aus:

prevFrame oder nextFrame stehen grundsätzlich für Goto and Stop. Um den Film weiter laufen zu lassen, kann man dem jeweiligen Befehl noch ein Play nachrücken. Zuerst werden alle Befehle eines Frames bearbeitet, bevor zu einem anderen Frame gesprungen wird. Ausnahmen bilden Call-Aktionen und Funktionsaufrufe.

```
Frame 2 :
    ifFrameLoaded ("Scene 1", 2000) {
        gotoAndPlay ("Scene 1", 4);
    }
```

```
Frame 3 :
prevFrame ();
play ();
```

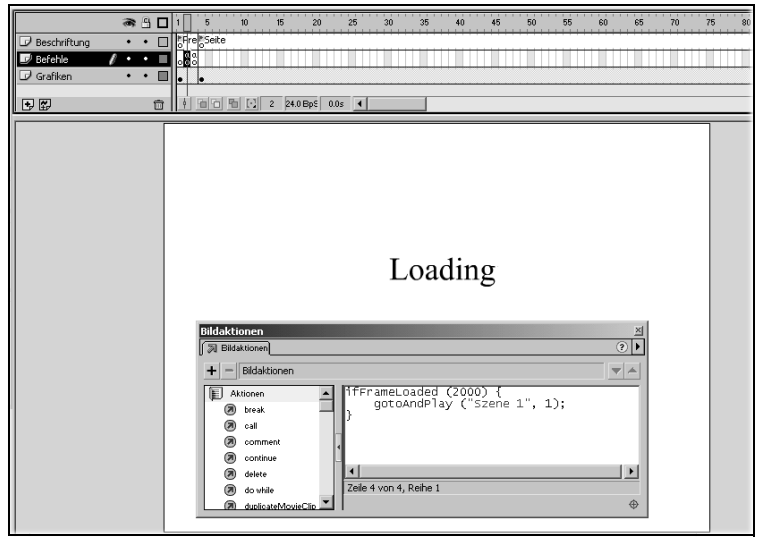
Der Preloader wird direkt zu Beginn der ersten Szene eingebaut, damit er auch sofort eingeladen wird.

```
ifFrameLoaded ("Scene 1", 2000) {
        gotoAndPlay ("Scene 1", 4);
    }
```

Flash überprüft, ob der 2oooste Frame eingeladen ist und springt dann – so dies zutrifft – zu Szene 1, Frame 4.

Auf der CD finden Sie zu allen Preloadern passende Beispiele.

Sie können dem Besucher jetzt anzeigen, dass die Seite geladen wird und dafür ein kleines Movie abspielen, um die Zeit des Ladevorganges für den Besucher zu verkürzen. Die Betonung liegt hier aber eindeutig auf klein. Wenn Sie hinterher einen Preloader haben der auch 100 kB umfasst, macht das Ganze keinen Sinn mehr. Eine Loadinganzeige entwerfen Sie am geschicktesten, indem Sie ein zweites Layer erzeugen und darin ein Movie erstellen, in dem z.B. der Schriftzug *Loading* erscheint. Diesen Schriftzug können Sie animieren, damit es nicht ganz so monoton aussieht.



▶ Preloader für mehrere Szenen

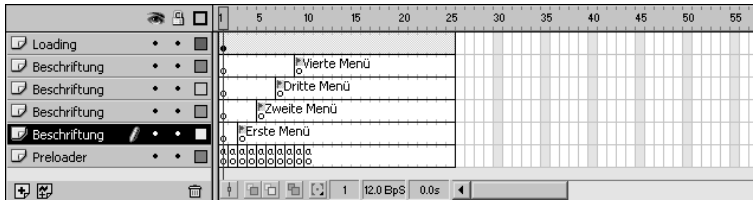
Wenn Sie mehrere Szenen einladen möchten – etwa die Untermenüs einer Website – nützt der bisherige Standard-Preloader wenig.

Ordner preloade/szenen, Datei: szenenpr.fla

```
ifFrameLoaded ("Scene 1", 2000) {
        gotoAndPlay ("Scene 1", 3);
    }
```

Der Standard-Preloader überprüft nur, ob die erste Szene eingeladen ist und überprüft nicht den Ladestatus der anderen Szenen. Sie könnten die anderen Szenen auch mit einem Standard-Preloader aufbauen und direkt an den Anfang der Szenen diesen Preloader setzten. Jedoch taucht dann das Problem auf, dass der Besucher eventuell zu einer Szene springt, in der dieser Preloader noch nicht eingeladen ist, so dass dann das SWF wieder „abstürzt".

Deshalb ist es ratsam, alle Preloader in die erste Szene einzubauen und die eigentliche Hauptseite erst in die zweite Szene zu stellen.



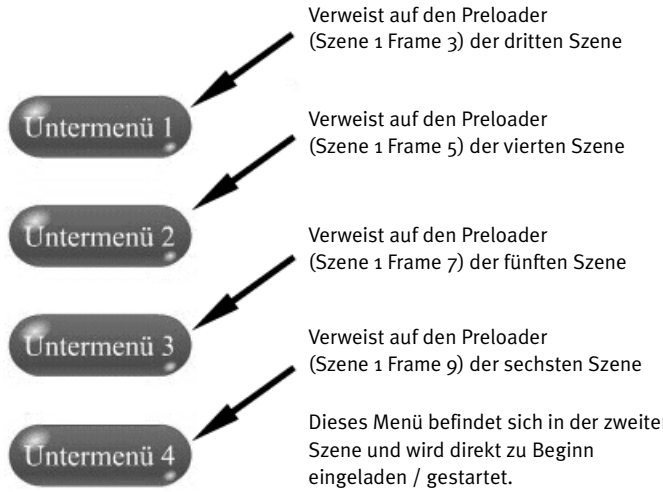
Im Prinzip ändert sich dadurch an den Preloadern wenig. Ein Unterschied liegt darin, dass der erste Preloader sofort den Ladestatus der zweiten Szene überprüft, da die erste Szene nur die Preloader enthält und deshalb schnell eingeladen ist. Vom Aufbau her sieht dies wie folgt aus:

```
Frame 1 :
    ifFrameLoaded ("Scene 2", 2000) {
        gotoAndPlay ("Scene 2", 1);
    }
Frame 2 :
prevFrame ();
play ();
Frame 3 :
    ifFrameLoaded ("Scene 3", 2000) {
        gotoAndPlay ("Scene ", 1);
    }

Frame 4 :
prevFrame ();
play ();

...
```

Die Buttons zum ersten oder zweiten Untermenü verweisen nicht direkt auf die Szenen, sondern rufen in der ersten Szene den jeweiligen Preloader auf.



Hier zum Beispiel das ActionScript des Buttons, um zum zweiten Untermenü zu gelangen:

```
On (release){
gotoAndPlay ("Scene 1", 3);
}
```

▶ Preloader für einzelne SWFs

Ordner preloade/einzSWF, Datei: haupt.fla

Leider hat auch der *Preloader für mehrere Szenen* einen Nachteil. Normalerweise kann man ihn für gängig aufgebaute Websites einsetzen, nur bei Websites mit extrem großen Untermenüs kann es Probleme mit der Ladezeit geben: Wenn der Besucher z.B. auf der Hauptseite angelangt ist und sofort zum fünften Untermenü bzw zur fünften Szene springen möchte, muss er warten, bis das zweite, dritte und vierte Untermenü eingeladen ist.

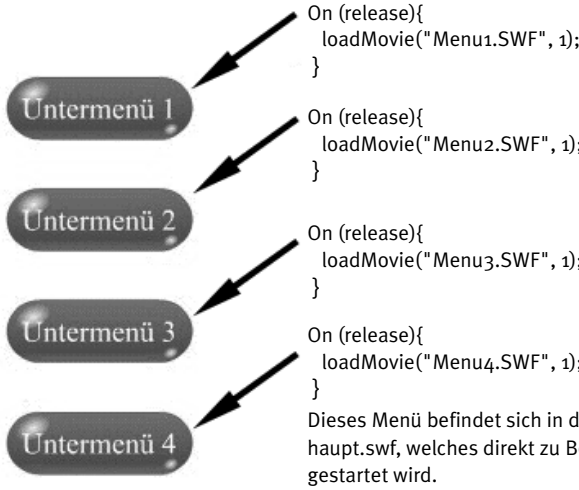
Wenn die Untermenüs bzw. Szenen nicht all zu groß sind, stellt dies kein Problem dar. Wenn allerdings jedes Untermenü um die 500 kB hat, ist es unter Umständen günstiger, die Untermenüs als einzelne SWFs zu erstellen. Natürlich kann man die beiden Lösungen auch kombinieren, also nur bestimmte Untermenüs in eigene SWFs „auslagern".

In dem folgendem Beispiel werden alle SWFs (Untermenüs) einzeln eingeladen.

Das erste SWF, das eingeladen wird, ist die Hauptseite mit einem einfachen *Standard-Preloader*. Die anderen einzelnen SWFs werden mit Buttons auf der Hauptseite mit dem Befehl

```
On (release){
loadMovie ("Ihr.SWF", 1);
}
```

eingebunden. Sobald der Besucher nun auf einen Button klickt, wird das entsprechende SWF in Level 1 geladen. In die einzelnen SWFs wird auch wieder der Standard-Preloader eingebaut.



Wenn Sie allerdings ein SWF in Level 1 einladen, sind in den nicht überdeckten Bereichen noch die Grafiken des im Hintergrund stehenden Levels 0 (Hauptseite) zu erkennen. Damit diese Grafiken nicht mehr im Hintergrund stören, gibt es mehrere Möglichkeiten:

Zum Aufbau von Levels
→ *Kapitel 1: Einführung.*

Am einfachsten ist es, in alle geladenen SWFs im untersten Layer ein flächenfüllendes Rechteck einzufügen. Diesem geben Sie dann die Farbe des Hintergrundes. Das Rechteck muss mindestens so groß wie die Bühne sein, da sonst nicht alle Stellen abgedeckt werden. Dieses Rechteck auf Level 1 verdeckt dann den Hauptlevel, der im Hintergrund liegt.

Die zweite Möglichkeit besteht darin, dem Button, der die anderen Untermenüs (andere SWFs) einlädt,

```
On (release){
loadMovie ("Ihr.SWF", 1);
}
```

noch einen Befehl hinzuzufügen, der die Hauptseite zu einem leeren Frame springen lässt:

```
On (release){
GotoAndStop (1999);
loadMovie ("Ihr.SWF", 1);
}
```

Ein einheitlicher Aufbau aller SWF-Dateien erleichert die Arbeit. Nutzen Sie die ersten drei Frames nur für den Preloader, wobei der erste Frame leer bleibt. So können Sie jedem Movieclip gotoandstop (1); zuweisen und damit sicherstellen, dass der Clip zu einem leeren Frame springt.

Voraussetzung ist natürlich, dass Frame 1999 existiert und leer ist.

Wenn das SWF in Level 1 eingeladen wurde, kann dieses mit dem Befehl `unloadMovie (1);` wieder entladen werden, so dass der Besucher zurück auf die Hauptseite kommt. Falls die Hauptseite, wie im zweiten Beispiel beschrieben, zu einem leeren Frame gesprungen ist, muss dieser gleichzeitig mit dem Unload-Befehl des ersten Levels wieder zum Startframe springen.

Um dem Besucher die Möglichkeit zu geben, wieder zur Hauptseite zurück zu kommen, können Sie in einem dieser SWFs einen Schalter platzieren, dem das folgende ActionScript zugewiesen ist:

```
unloadMovie (1);
tellTarget ("_level0") {
      gotoAndPlay (3);
}
```

Eine andere Möglichkeit besteht darin, die Untermenüs nicht in Level 1, sondern direkt in Level 0 einzuladen. Auf diese Weisen müssen Sie nicht jedes mal den Hauptlevel 0 zu einem leeren Frame springen lassen, da dieser überschrieben wird. Wenn Sie allerdings einen Film in Level 0 neu einladen, werden alle anderen Level automatisch mitgelöscht werden, da Level 0 der Hauptlevel ist. Wenn zum Beispiel Level 5 die Musik enthält (in Flash wird die Hintergrundmusik in der Regel in ein eigenes Level eingeladen) und dann in Level 0 ein neues SWF geladen wird, wird automatisch Level 5 entladen. Zudem verlieren Sie durch das „Überschreiben" des ersten Levels, auch alle Variablen, die Sie bisher gesetzt haben.

Außerdem ist es von Vorteil, die Untermenüs in Level 1 einzuladen. So könnte man die Untermenüs so anlegen, dass Sie nicht die gesamte Fläche abdecken und die Navigation des Hauptlevels noch mit angezeigt wird, so dass dies wie bei einem Frameset aussähe.

Sicherlich gibt es noch andere Gestaltungsmöglichkeiten. Es würde zu weit führen, dies alles an dieser Stelle zu beschreiben. An sich reicht es aus, eine Methode gut zu kennen, da alle Varianten letztendlich zum selben Ergebnis führen.

Kommen wir nun wieder zum Anfang zurück – zum Aufbau einer Site mit einzelnen SWFs. Diese Vorgehensweise hat einen ganz entscheidenden Nachteil: Die SWFs werden erst eingeladen, wenn der Besucher versucht, das Untermenü anzuwählen. Auch wenn er auf der Hauptseite verweilt, werden die einzelnen SWFs noch nicht eingeladen.

▶ Preloader für mehrere SWFs

Ordner preloade/mehrSWF, Datei: preloade.fla und preload2.fla

Die einzelnen SWFs für die Untermenüs kann man automatisch einladen lassen, indem man einen separaten Preloader für alle einzuladenden Daten/Files in einem einzelnen SWF baut. Dieser ist der Hauptseite vorgeschaltet und wird so als Erstes geladen.

Bevor wir auf den Aufbau und die Funktionsweise dieses Preloaders eingehen, einige Worte vorweg: Diese Variante des Preloaders ist sehr umständlich zu handhaben, Fehler lassen sich nur schwer lokalisieren und der Arbeitsaufwand ist vergleichsweise hoch. Sie sollen aber bei dem Beschriebenen nicht nur den Preloader sehen, sondern auch verstehen, wie man mit Flash Variablen zwischen verschiedenen Filmen austauschen kann bzw. wie man die `if-frames-loaded`-Befehle zu einem komplexen Block zusammenfasst. In Einzelfällen kann man diesen Preloader allerdings gut benutzen, z. B. zum Einladen mehrerer längerer Musikstücke.

Grundprinzip:
Falls der Besucher auf der Hauptseite bleibt und Flash trotzdem die restlichen Untermenüs einladen soll, ist dafür eine komplizierte Verkettung von Variablen und Abfragen notwendig. Wenn der Besucher zunächst auf der Hauptseite verweilt und der Preloader beginnt, das erste Untermenü einzuladen, der Besucher sich dann aber entschließt, in den dritten Untermenüpunkt zu gehen, muss der Preloader den aktuellen Ladevorgang abrechen, überprüfen ob das dritte Untermenü eingeladen ist, um dieses zu starten bzw. dieses einzuladen und anschließend zu starten. Alleine diese Beschreibung zeigt, wie vielfältig der Preloader sein muss, da es hier um das Einladen mehrerer Dateien geht, wobei die Ladepriorität wechseln kann.

Aufbau:
Wie schon weiter oben erwähnt, wird als erstes ein SWF gestartet, aus dem alle anderen SWFs eingeladen werden: also ein SWF extra für die Preloader. Dieses SWF kann auf diese Weise durchaus über 50 kB groß werden. Deshalb muss dort auch im ersten Frame das Einladen des kompletten SWFs überprüft werden. Dafür kann man einen Standardpreloader benutzen. Dieser springt dann allerdings nicht zum vierten Frame, sondern gleich zu Frame 9, wobei die Belegung der Frames natürlich frei wählbar ist. In der Beispieldatei sind am Anfang noch ungenutzte Frames vorhanden, die überflüssig wirken, sich aber als nützlich erweisen, wenn der Preloader noch erweitert oder umgebaut werden soll. Die leeren Frames können Sie also als Reserve-Frames betrachten.
Es gibt einige grundlegende Variablen bei diesen Preloader:

`load`, `loadIn` und `play` (`play` ist hierbei auch ein Variablenname !)

`load` wird benutzt, um dem Besucher der Seite anzuzeigen, welche Datei bzw. welcher Seitenteil gerade eingeladen wird. Der Movieclip besitzt ein Textfeld namens `load`, so dass der Variablen `load` immer ein String zugewiesen wird.

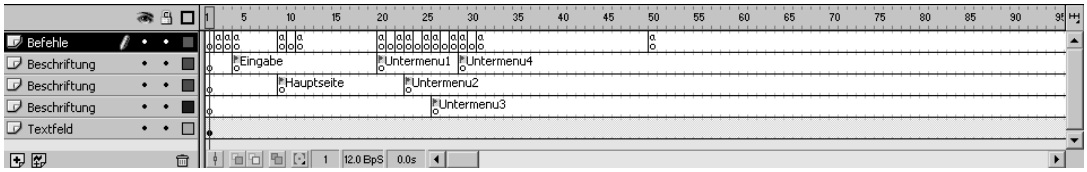
`play` ist eine Variable für den Preloader, sie enthält die Nummer des entsprechenden Frames, in dem der Preloader für das zu startende Movie sitzt. Normalerweise lädt der Preloader kontinuierlich von links nach rechts die eingegebenen SWFs ein. Wenn der Besucher sich aber entschließt, zuerst

z.B. in das fünfte Untermenü hineinzusehen, teilt die Variable `play` dem Preloader mit, dass dieser zu dem Frame springen soll, in dem dann das SWF für das fünfte Untermenü eingeladen wird.

Die Variable `loadIn` enthält die Nummer des Levels, in die gerade ein Movie eingeladen wird – der Preloader benötigt die Levelzahl, um das Movie zu entladen, falls der Ladevorgang abgebrochen werden muss.

Zudem benötigt man für jeden Movieclip noch eine eigene Variable, um zu kennzeichnen, ob dieser schon eingeladen ist oder nicht.

In dem Beispiel wurden die Variablen `Menu1`, `Menu2` usw. gewählt.



Der Aufbau ist auch gegliedert.

Theoretisch können Sie mit diesem Preloader beliebig viele SWF-Dateien einladen. Allerdings muss man in etwa mit 1 kB belegtem Speicherplatz pro einzuladendem SWF rechnen. Wird der Preloader zu groß, können Sie die SWF-Dateien auf mehrere Preloader aufteilen und diese nacheinander starten.

Frame 1 bis 3 beinhaltet den Standard-Preloader für das SWF mit den Preloadern.

Das vierte Frame wird benutzt, um den augenblicklichen Ladevorgang abzubrechen und ein anderes SWF einzuladen.

Frame 9 bis 19 enthalten die SWfs, die zum eigentlichen Start der Seite benötigt werden. Mit dem Einladen dieser Dateien wird unverzüglich nach dem Start der SWFs mit den Preloadern begonnen. Hier sollten alle SWF-Dateien aufgelistet sein, die vor dem Start der Hauptseite eingeladen sein müssen.

Ab Frame 20 werden alle einzuladenden Dateien nach demselben Schema (für jedes SWF drei Frames) aufgelistet.

Wenn der Preloader startet, überprüft er zuerst, ob er selbst komplett eingeladen ist. Falls dies der Fall ist, springt er direkt zu den Frames mit den Ladeanweisungen für die anderen SWFs, in diesem Beispiel zum neunten Frame.

Frame 2:

```
load = "Preloader";
loadin = "0";
ifFrameLoaded (_totalframes) {
    play = 9;
    gotoAndPlay (9);
}
```

Frame 3:

```
gotoAndPlay (2);
```

Im zweiten Frame wird zuerst `load = "Preloader"` gesetzt. Dadurch erscheint in dem Textfeld namens `load` der Schriftzug „Preloader“. Das dem Textfeld voranstehende „Lade“ teilt dem Besucher mit, wenn der Preloader eingeladen wird. Dies steht dort im Normalfall aber nicht lange, da der Preloader meist sehr klein und daher schnell eingeladen ist.

`Loadin = 0;` sagt aus, dass der Preloader gerade in Level 0 eingeladen wird. Danach folgt die eigentliche Abfrage, ob der Preloader schon eingeladen ist.

```
ifFrameLoaded (_totalframes) {
```

Hier wird überprüft, ob alle Frames des Preloaders schon eingeladen sind. Falls dies zutrifft, wird die Variable `play = 9` gesetzt und zu Frame 9 gesprungen, da im neunten Frame die Ladeanweisung für die Hauptseite steht.

Achten Sie darauf, der Variable `Play` den richtigen Wert zuzuweisen, damit das richtige SWF eingeladen bzw. gestartet wird. Dies ist einer der Nachteile dieses Preloaders. Es empfiehlt sich daher, zu Beginn eine kleine Tabelle mit den einzuladenden SWF-Dateien, den dazu gehörigen `Playwerten` und dem jeweiligen Level zu erstellen.

Name des SWF	Playwerte *	Variablenname	Level
preloader.swf	----	---	0
haupt.swf	9	haupt	5
menu1.swf	20	menu1	10
menu2.swf	23	menu2	11
menu3.swf	26	menu3	12
menu4.swf	29	menu4	13

Frameposition mit den Ladeanweisungen im Preloader

Man erkennt, dass zwischen den verschiedenen Levels noch Platz ist, so sind z.B. die Level 1–4 und 6–9 unbenutzt.

Der Vorteil der nicht benutzen Levels fällt Ihnen spätestens dann auf, wenn Sie Ihre Seite erweitern wollen, aber nicht in höhere Levels ausweichen können, da Ihre Verbesserungen im Hintergrund unter den anderen Levels liegen müssen.

Wie schon erwähnt springt der Preloader zum neunten Frame, nachdem er komplett eingeladen wurde. Dort wird zuerst geprüft, ob die Hauptseite schon geladen ist. Das geschieht anhand der Variable `haupt`. Sie enthält den Wert 1, wenn die Hauptseite schon eingeladen ist. Diese Variable wird aus dem jeweiligen einzuladenden SWF gesetzt. Jedes einzuladende SWF enthält einen Standardpreloader, der, wenn der komplette Film eingeladen ist, die entsprechende Variable auf der Hauptbühne den Wert 1 zuweist und anschließend selbst zu Frame 1 springt und dort stoppt. Da der erste Frame bei allen einzuladenden SWF-Dateien leer ist, merkt der Besucher nicht, dass eine SWF-Datei im Hintergrund eingeladen wurde.

Frame 9: Anstatt eine Variable auf 1 bzw. 0 zu setzen, gibt es auch die Möglichkeit, diese auf „true“ bzw. „false“ zu setzen, wobei die Methode mit den Booleanwerten weniger Arbeitspeicher benötigt.

```
if (haupt == 1) {
    gotoAndStop (_currentframe+2);
} else {
    loadMovieNum ("haupt.swf",5);
    loadin = "5";
    load = "Hauptseite";
    play ();
}
```

Im neunten Frame wird nur geprüft, ob das jeweilige SWF schon eingeladen wurde. Falls dies noch nicht geschehen ist, wird es in den entsprechenden Level eingeladen und der Preloader springt weiter zu den nächsten Frames (hier zu Frame 11), in denen der Ladevorgang des SWFs überprüft wird.

Frame 11:

```
if (haupt == 1) {
    loadin = "0";
    load = "--------------------------";
    if (play == 9) {
        _level5.gotoAndStop(4);
    }
    gotoAndPlay (20);
} else {
    prevFrame ();
}
play ();
```

Zuerst wird überprüft, ob `haupt` gleich 1 ist. Falls dies zutrifft, springt der Preloader sofort zu den Frames mit den nächsten Ladeanweisungen. [Hier: Frame 20].

Frameloop
→ *Kapitel 3: Begriffserklärung.*

Falls dies nicht zutrifft, springt der Preloader zu dem vorherigen Frame und wiederholt dieses so lange, bis die Hauptseite vollständig eingeladen ist. Diese zwei Frames (hier 10 und 11) werden also als Frameloop abgespielt. Wenn die Hauptseite vollständig eingeladen wurde, wird überprüft, ob die Variable `play` den Wert 9 hat. Falls dies zutrifft, wird Level 5 und somit unser gerade eingeladenes SWF, abgespielt.

Die restlichen Frames folgen dem selben Aufbauschema: jeweils drei Frames pro einzuladendes SWF. Zuerst die Ladeanweisung für den Fall, dass es noch nicht eingeladen wurde, danach ein leerer Frame für das Frameloop und schließlich die Abfrage, ob das SWF komplett eingeladen wurde und ob es gestartet werden soll.

Wenn man allerdings den dritten Frame (Frame 11) mit einem der späteren dritten Frames (Frame 22, 25, 28...) vergleicht, entdeckt man kleine Unterschiede:

Frame 22:

```
if (Menu1 == 1) {
    loadin = "0";
    load = "--------------------------";
    if (play == 20) {
        _level10.gotoAndStop(4);
        gotoAndPlay (9);
        play = 0;
    }
} else {
    prevFrame ();
}
play ();
```

Frame 2 unterscheidet sich von Frame 9 durch die sechste und siebte Zeile

```
gotoAndPlay (9);
play = 0;
```

Diese Befehlszeilen findet man in Frame 11 nicht. Benötigt werden sie bei den späteren Frames: Wenn ein Menü/SWF auf Anweisung des Besuchers geladen wird, ein davorliegendes Menü aber noch nicht eingeladen wurde, würde der Preloader das Einladen des Menüpunktes abschließen und dann in Ruhestellung gehen (mit `stop` in Frame 50). Dies ist aber nicht gewünscht, da der Preloader alle Untermenüs im Hintergrund einladen soll.

Deshalb stehen dort diese zwei Befehlszeilen. Die erste bewirkt, dass der Preloader wieder zum Anfang springt und noch einmal überprüft, ob alle SWF–Dateien eingeladen sind. Die zweite Befehlszeile dient dazu, dass das gerade gestartete SWF nicht noch einmal durch die Variable `Play` neu gestartet wird.

Falls man eine aufwendigere Seite mit mehreren Menü- und Untermenü-Ebenen aufbaut, könnte man diese Zeile

```
gotoAndPlay (9);
```

so abändern, dass nicht wieder zum Anfang gesprungen wird, sondern zu der Stelle, an der die Untermenüs des erstens Untermenüs geladen werden, um so dasjenige SWF (Untermenü) vorzuziehen, dass der Besucher vermutlich als nächstes ansteuern wird.

Frame 4:

```
load = "Abgebrochen";
if (loadin ne 0) {
    unloadMovieNum (loadin);
}
gotoAndPlay (play);
```

Wenn Frame 4 abgespielt wird, wird der augenblickliche Ladevorgang abgebrochen und ein neues SWF gestartet. Zuerst wird dem Besucher der Seite ein „Lade: Abgebrochen“ im Textfeld angezeigt.

Die Variable `loadIn` enthält die Nummer des Levels, in den gerade ein SWF eingeladen wird. Durch `unloadMovieNum (loadin);` wird der entsprechende Level dann entladen (die bisher geladenen Daten des SWF verbleiben im Cache des jeweiligen Besuchers). Nachdem dafür gesorgt wurde, dass kein SWF mehr eingeladen wird, springt der Preloader zu den Frames mit den Lade- und Start-Anweisungen des jeweiligen Untermenüs (SWF), welches gestartet werden soll. Die entsprechende Nummer des Frames mit den Lade und Startanweisungen übermittelt die Variable `play`.

▶ Zusätze für Preloader

_totalframes-Eigenschaft • Die totalframes-Eigenschaft gibt aus, wie viele Frames jeweilige SWF enhält. Diese kann man z.B. in den Standardpreloader miteinbauen, so dass man den Preloader dann überall einsetzen kann.

```
if (_framesloaded >=_totalframes)
   {
    gotoAndPlay ("Scene 1", 4);
   }
```

Für die Benutzung von `_totalframes` im Zusammenhang mit Szenen ist jedoch noch eine Kleinigkeit wichtig. Flash setzt die Szenen einfach aneinander. Wenn man einen Film mit mehreren Szenen erstellt, in dem kein ActionScript enthalten ist, wird zuerst die erste Szene komplett abgespielt, dann die zweite usw.

Flash behandelt die Szenen wie einen großen Film. Flash unterteilt die Szenen also nicht richtig, sondern zeigt diese nur unterteilt und übersichtlicher an. Für Flash sind Szenen nichts anderes als hintereinander liegende Frames.

Wenn man die _totalframes-Eigenschaft auf einen Film mit mehreren Szenen anwendet, gibt Flash also nicht die Anzahl der Frames der jeweiligen Szene wieder, sondern die des ganzen Films (alle Frames in allen Szenen). Szenen sind hierbei nicht zu verwechseln mit eingebetteten Movies, diese sind nicht in der _totalframes-Abfrage beinhaltet, da es sich um eigenständige Filme handelt.

`_totalframes` gibt die gesamte Anzahl der Frames innerhalb eines SWFs an, egal ob diese durch Szenen getrennt sind oder nicht.

Für den Preloader für mehrere Szenen kann man deshalb nicht die _totalframes-Abfrage verwenden. In diesem Fall würde man den ganzen Film preloaden und könnte nicht jede Szene einzeln abfragen.

→ *Szenen-Preloader.*

Der Vorteil der Einzelabfrage ist, dass beim Einladen dem Besucher schon früher die Startseite präsentiert werden kann. Angenommen eine Internetseite hat fünf Menüpunkte: Die eigentliche Hauptseite (Szene 2) ist

nur 70 kB groß, aber die dahinter liegenden Seiten haben aufgrund vieler Grafiken eine Größe von 1000 kB. Das heißt, die Seite hat eine Gesamtgröße von 1070 kB. Würde man einen Preloader für die gesamte Seite vorsehen, würde der Ladevorgang viel zu viel Zeit beanspruchen. Wenn Sie hingegen für jede einzelne Szene einen eigenen Preloader einsetzen, bekommt der Besucher fast sofort die Hauptseite angezeigt und währenddessen werden im Hintergrund die übrigen Untermenüs/Szenen eingeladen.

_framesloaded-Eigenschaft • Die ifFrameLoaded-Abfrage wurde weiter oben schon erläutert. Mit der _framesloaded-Eigenschaft ist es nun aber auch möglich, diese mit einem If-Befehl zu kombinieren.

Statt

```
ifFrameLoaded ("Scene 1", 2000) {
    gotoAndPlay ("Scene 1", 3);
}
```

geht auch

```
if (_framesloaded >=_2000)
   {
    gotoAndPlay ("Scene 1", 3);
   }
```

Der Nachteil ist, dass man immer nur komplette Movies abfragen kann. Die Unterscheidung verschiedener Szenen ist hiermit nicht mehr möglich. Verschiedene Movies kann man mit Hilfe der Punkt-Syntax ansprechen.

Punktsyntax
→ *Kapitel 1: Einführung und Kapitel 2: Neuerungen in Flash 5.*

Diesen Befehl kann man zum Beispiel auch dazu benutzen, um abzufragen, ob ein bestimmter Frame schon eingeladen wurde. Sollte dies zutreffen, kann man die davor liegendenden Frames abspielen lassen,während im Hintergrund die restliche Hauptseite eingeladen wird.

Möglich ist auch, einen Else-Befehl mit einzubinden bzw. eine Verschachtelung mit anderen Befehlen, z.B. im Zusammenhang mit der _totalframes-Eigenschaft

Mit den Eigenschaften `_frameloaded` und `_totalframes` könnte man auch eine prozentuale Darstellung erzeugen.

```
ProzentAusgabe= int((_level0._framesloaded/
_level0._totalframes)*100);
balken.gotoAndStop(ProzentAusgabe);
if ( _root._framesloaded == _root._totalframes ) {
_root.gotoAndPlay(2);
} else {
_root.gotoAndStop(1);
}
```

Ordner preloade/frameslo, Datei: framelo.fla

Da bei der Berechnung der Prozentzahl die Anzahl der eingeladenen Frames zugrundegelegt wird, funktioniert die Prozentanzeige leider nur dann genau, wenn alle Frames in etwa die gleiche Datenmenge beinhalten. Wenn beispielsweise der Film aus 100 Frames besteht und der 100ste Frame eine große Grafikdatei enthält, die vorher nicht vorkam, würde der Preloader sehr schnell anzeigen, dass die ersten 99 % eingeladen wären und dann lange bei 99 % stehen bleiben, um diese Grafik einzuladen.

Prozentuale Ladeanzeige mit Hilfe von getBytesLoaded() und getBytesTotal() • Da eine exakte prozentuale Ladeanzeige mit `_framesloaded` und `_totalframes` nicht möglich ist, muss man für eine genaue Prozentangabe die bereits eingeladenen Bytes und die Gesamtdateigröße abfragen. Dies geht mit den Eigenschaften `getBytesLoaded()` und `getBytesTotal()`.

Ordner preloade/prozent, Datei: prozent.fla

```
insgesamt = (int(_root.getBytesTotal()/10.24))/100;
geladen = (int(_root.getBytesLoaded()/10.24))/100;
ProzentAusgabe = (_root.getBytesLoaded()/_root.getBytes-
Total())*100;
if (balken._currentframe<ProzentAusgabe) {
    balken.play();
} else {
    balken.stop();
}
if (_root.getBytesLoaded() == _root.getBytesTotal()) {
    _root.gotoAndPlay(2);
} else {
    _root.gotoAndStop(1);
}
verbleibend = insgesamt-geladen+" kB";
insgesamt = insgesamt+" kB";
geladen = geladen+" kB";
ProzentAusgabe = ProzentAusgabe + " %";
```

Als erstes wird die Gesamtgröße des SWFs abgefragt. Die Daten werden in Bytes eingelesen. Um dem Besucher den Wert in kB anzuzeigen, muss man die Anzahl der Bytes durch 1024 teilen (1Byte = 8 Bit).

Soll das Ergebnis gerundet werden, so steht hierfür der Befehl `Int()` zur Verfügung. Mit ihm kann man allerdings nur eine Kommazahl auf eine ganze Zahl abrunden kann. Um z. B. auf zwei Stellen hinter dem Komma zu runden, muss man zuerst die Zahl durch 1024 teilen und mit 100 multiplizieren (dies entspricht einer Teilung durch 10.24), das Ergebnis dann mit `Int()` runden und anschließend die Multiplikation durch eine Division mit 100 rückgängig machen.

Danach erfolgt genau die gleiche Berechnung mit den bisher eingeladenen Bytes.

Nachdem man beide Werte eingelesen hat, kann man mit ihnen analog den prozentualen Anteil der bisher geladenen Daten, die Gesamtgröße des SWF-Files sowie die Menge der bereits eingeladenen und der noch nicht eingeladenen Daten ermitteln.

```
if (balken._currentframe<ProzentAusgabe) {
    balken.play();
} else {
    balken.stop();
}
```

Diese If-Schleife überprüft, in welchem Frame sich das Movie *Balken* befindet. Das Movie *Balken* besteht genau aus 100 Frames, so dass die Anzahl der Frames der Prozentzahl entspricht. Die If-Abfrage mit den Play- und Stop-Befehlen sorgt dafür, dass das Movie *Balken* immer bis zu der richtigen Stelle abspielt und dort stehen bleibt (entsprechend der eingeladenen Prozentzahl). Man hätte auch direkt `balken.gotoAndStop (Prozentausgabe);` schreiben können. Dies würde aber zur Folge haben, dass der Balken den Ladestatus anzeigt, indem er ohne langsame Animation sofort zu dem jeweiligen Frame springt.

```
if (_root.getBytesLoaded() == _root.getBytesTotal()) {
    _root.gotoAndPlay(2);
} else {
    _root.gotoAndStop(1);
}
```

Die darauf folgende If-Abfrage ist zum Stoppen des Hauptfilmes gedacht. Da dieser beim Abspielen des Preloaders gerade eingeladen wird. Wenn der Ladevorgang beendet wurde, wird durch die If-Abfrage der Hauptfilm weiter abgespielt.

```
verbleibend = insgesamt-geladen+" kB";
insgesamt = insgesamt+" kB";
geladen = geladen+" kB";
```

ProzentAusgabe = ProzentAusgabe + " %";

Nach der If-Abfrage folgen nur noch Zuweisungen der Werte, damit die Ausgabe in die Textfelder vollständiger aussieht. Den ermittelten Werten werden hier die Einheiten kB bzw. % angehängt.

Frameübergreifende Steuerung

Dieser Abschnitt erläutert die Realisierung einer frameübergreifenden Steuerung. Möchten Sie Werte oder Kommandos an einen anderen Frame übermitteln oder einen Flash-Film in einem anderen Frame steuern, so realisieren Sie dieses am besten mit einem JavaScript. Ein Flash-Film kann mittels FS Command aus einer Seite angesprochen werden und dieses wiederum arbeitet mit JavaScript zusammen. Frames werden im Frameset, also

der übergeordneten Seite, Namen zugeordnet, die Sie im Folgenden einheitlich verwenden müssen. Haben Sie erst einmal ein Script in Ihrer „Zielseite“, welches den Film von außen steuert, können Sie nun mit `GetURL` alle JavaScripte mit folgender Syntax von anderen Seiten aus aufrufen:

```
javascript:parent.frames.FRAMENAME.FSKommando() ;
```

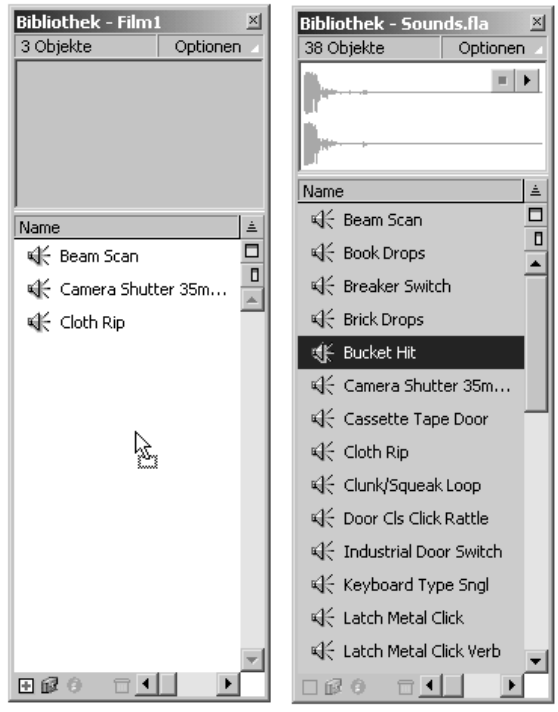
`FRAMENAME` entspricht hier der Namensgebung der Zielseite im Frameset und `FSKommando()` ist das auf der entsprechenden Seite enthaltene Script.

Sound

Mit Flash ist es möglich, verschiedene Aktionen mit Sound zu unterlegen. Äuch längere Musikstücke kann man ohne Probleme im Hintergrund laufen lassen. Wie man dies am besten in Szene setzt, wird im Folgenden beschrieben.

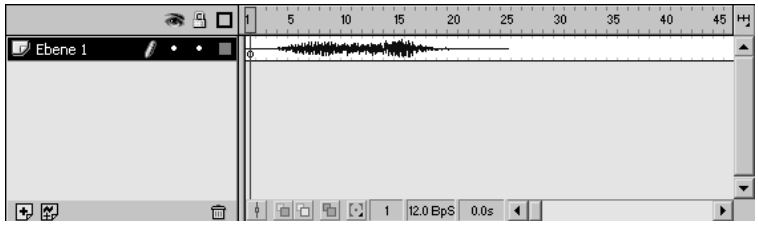
Die Bibliothek Ihres Films können Sie mit STRG + L *aufrufen (L steht für Library).*

Flash 5 bringt eine Menge an Soundsamples mit. Sie finden sie UNTER FENSTER › ALLGEMEINE BIBLIOTHEKEN › SOUNDS. Per Drag and Drop können Sie die Samples der *Allgemeinen Bibliothek* in die Bibliothek Ihres Films verschieben.

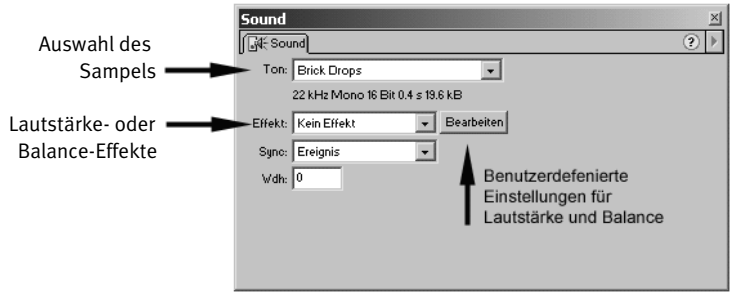


Sie können die Samples auch direkt auf die Bühne Ihres Films ziehen, dann werden diese direkt dem jeweiligen Frame zugewiesen und an dieser Stelle abgespielt. Ein anderes Soundsample importieren Sie über DATEI › IMPORTIEREN.

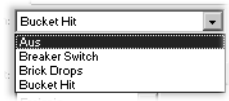
Ein einem Frame zugeordnetes Sample wird in der Frameleiste wie folgt angezeigt:



Über das Bedienfeld SOUND können Sie ein Soundsample einem Frame zuordnen bzw. dieses ändern. Sie können es mit einem Klick mit der rechten Maustaste auf das entsprechende Frame in der Frameleiste – unter BEDIENFELDER › SOUND – aufrufen.



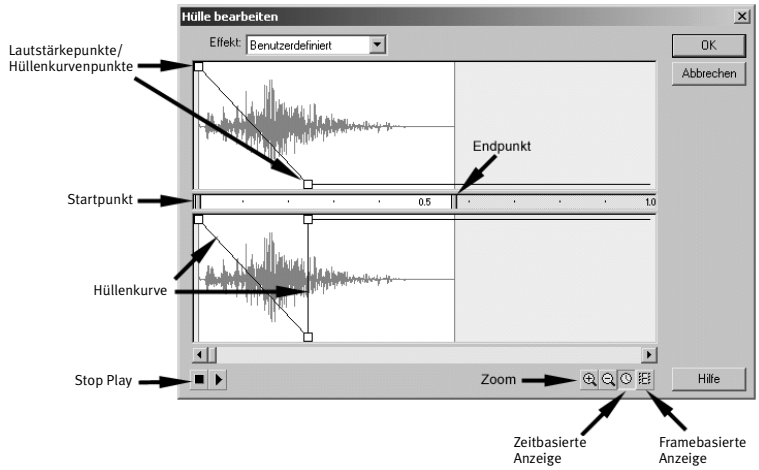
Unter dem Menü Ton können Sie eines der bisher importierten/eingefügten Soundsamples dem jeweiligen Frame zuordnen.



In dem daneben liegenden Feld *Effekt* können Sie die Eigenschaften des Sounds festlegen, z.B. ob dieser eingeblendet oder ausgeblendet werden soll.

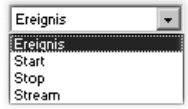
Indem Sie die Schaltfläche BEARBEITEN betätigen, können Sie die Soundeigenschaften selbst definieren. Den Lautstärkepegel für den rechten und linken Kanal können Sie variieren, so dass Sie eine eigenen Hüllenkurve für den jeweiligen Sound erstellen können.

Sound sollte so spät wie möglich importiert werden, da dieser den Testlauf zum Erstellen eines SFWs erheblich verlängert. Falls Sie den Sound vorab benötigen, sollten Sie die Qualität erst einmal auf die niedrigste Stufe setzen (Datei > Einstellungen für Veröffentlichung > Flash oder direkt in der Bibliothek Ihres Filmes unter „Eigenschaften“ des jeweiligen Sounds).



Wenn Sie unterschiedliche Hüllenkurven erzeugen, erhalten Sie unterschiedliche Soundvarianten. Auf diese Weise brauchen Sie eventuell keinen neuen Sound zu importieren und Ihr SWF wird kleiner.

Durch Klicken auf die Soundsamples erzeugen Sie neue Lautstärkepegelbegrenzer, die Sie beliebig, für jeden Kanal einzeln, verschieben können. Über die Start- und Stoppunkte können Sie einzelne Sequenzen wegschneiden, so dass nur bestimmte Teile im Loop laufen und immer nur ein Teil des Soundsamples abgespielt wird.



In dem Feld *Sync* können Sie das Abspielverhalten des Soundsamples einstellen.

- *Ereignis* bedeutet, dass der Sound an der Position komplett abgespielt wird.
- Mit *Start* wird das Soundsample hingegen nur so lange abgespielt, wie der Key-Frame, in dem sich das Soundsample befindet, nicht durch einen neuen Keyframe unterbrochen wird.
- Mit *Stop* wird das angegebene Soundsample gestoppt.
- *Stream*; ist für eine Veröffentlichung im Internet gedacht. Hier braucht nur der Anfang des Soundsamples eingeladen zu sein, um abgespielt zu werden. Der Sound wird immer passend zu den Frames abgespielt. Wenn die Frames nicht schnell genug eingeladen werden, werden sie übersprungen. Falls die Brandbreite des Users zu gering ist, um den Sound im voraus einzuladen, kann dies dazu führen, dass der Film kurz anhält.

In dem Feld *Wiederholungen* kann angegeben werden, wie oft das jeweilige Soundsample hintereinander abgespielt werden soll.

Die Größe des **Soundpuffers** für den Streaming-Sound kann man durch einen ActionScriptbefehl ändern. Standardmäßig steht dieser Soundpuffer auf 5 Sekunden. Das heißt, es müssen fünf Sekunden Laufzeit des Soundsamples eingeladen sein, bevor dieses abgespielt wird. Dieser Wert

kann durch den Befehl `_soundbuftime` geändert werden. Hier wird z. B. die Soundbuffertime auf 10 Sekunden erhöht:

```
_soundbuftime = 10;
```

Man sollte diesen Wert bei Sounds, die in sehr guter Qualität für die Übertragung bereit gestellt werden, erhöhen – ansonsten würde es beim Abruf durch Besucher mit geringen Datenübertragungsraten zu Aussetzern kommen.

Flash-Detection

Die wohl verbreitetste Variante der Flash-Detection sind so genannte Sniffer. Auf der Website befindet sich in diesem Fall ein Flashfilm, der nur die Funktion hat, auf die Seite mit dem eigentlichen Flash-Inhalt zu verlinken. Falls dies nicht erfolgt, kann davon ausgegangen werden, dass kein Plug-In vorhanden ist bzw. die Flash-Dateien nicht abgespielt werden können. Für diesen Fall baut man auf der ersten Seite eine automatische Weiterleitung nach 10 Sekunden ein, die auf die entsprechende Seite ohne Flash-Inhalte führt.

Grundlagen Smartclips

In der Grundversion von Flash 5 sind in der Allgemeinen Bibliothek Smartclips enthalten. Im Folgenden beschreiben wir, wie man sie einfügt, benutzt und anpasst.

Bei Smartclips handelt es sich um fertige Movies, die man in sein Projekt einfügen kann. Smartclips haben dabei den Vorteil, dass man ihnen Sequenzeigenschaften zuweisen kann, wodurch eine komfortablere Bedienung gewährleistet ist.

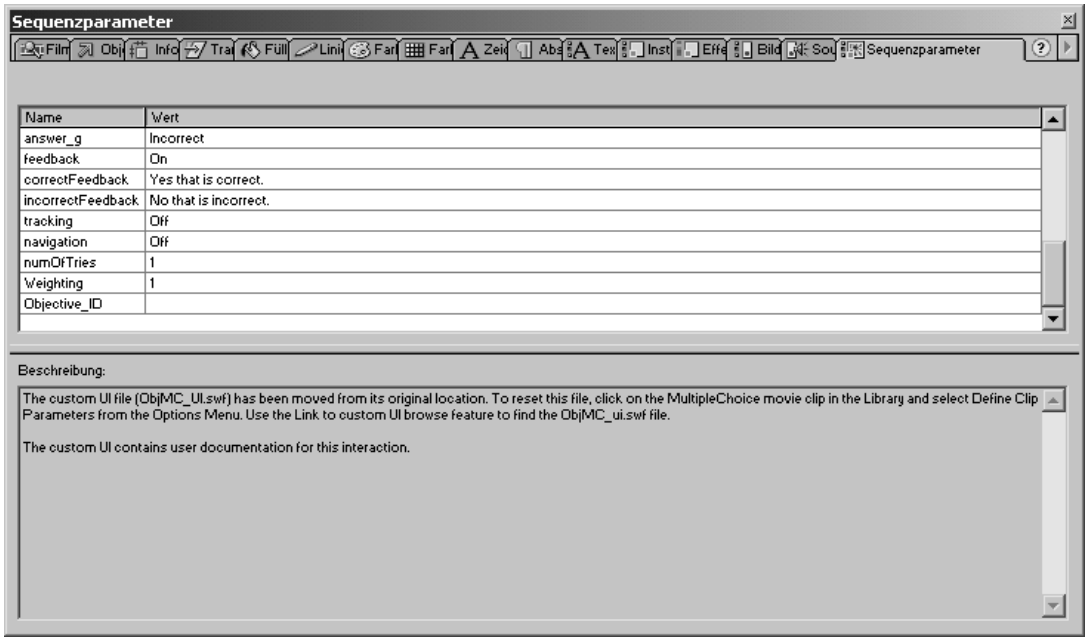
Wie man Smartclips selbst erstellt, wird in → 92 Erstellen von Smartclips beschrieben.

▶ Einfügen eines Smartclips

Smartclips befinden sich in der Bibliothek eines Filmes. Um sie zu nutzen, muss man die Smartclips aus der Bibliothek des anderen Filmes in die Bibliothek des eigenes Filmes kopieren. Dafür gibt es mehrere Möglichkeiten. Am einfachsten ist es, beide Filme und beide Bibliotheken zu öffnen und die Filmsequenz per Drag and Drop in die andere Bibliothek zu kopieren.

→ 74 Soundeffekte – Kopieren von Elementen aus einer Bibliothek.

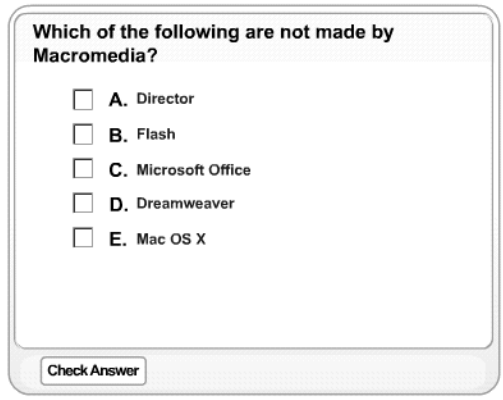
In dem Beispiel werden wir auf einen Smartclip aus der Allgemeinen Bibliothek eingehen (Fenster › Allgemeine Bibliothek › Lernende Objekte). Auch die Lernenden Objekte sind Smartclips, obwohl diese nicht bei den Smartclips einsortiert sind. Im Folgenden werden wir den Smartclip *MultipleChoice* beschreiben, der von Macromedia angefertigt wurde und dementsprechend umfangreich und komplex ist. Ein entsprechendes Movie, das zeigt, wie man einen Smartclip erstellt, wird auf im Abschnitt → *92 Erstellen von Smartclips* beschrieben.



▶ **Verändern der Sequenzparameter**

Ziehen Sie diesen Smartclip aus der Allgemeinen Bibliothek auf die Bühne. Jetzt können Sie es unter BEDIENFELDER › SEQUENZPARAMETER für den jeweiligen Zweck anpassen.

Wenn man das Movie startet, ohne die Sequenzparameter zu verändern, sieht dies folgendermaßen aus:



Wenn Sie nun die Fragestellung und die zur Auswahl stehenden Antworten ändern möchten, können Sie dies direkt über die Sequenzparameter machen, ohne in dem eigentlichen Movie Textfelder oder ActionScripts abändern zu müssen.

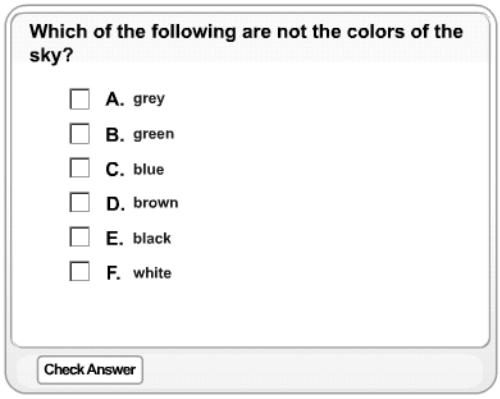
Die geänderten Sequenzparameter sehen dann folgendermaßen aus:

Name	Wert
question	Which of the following are not the colors of the sky?
choice_a	grey
choice_b	green
choice_c	blue
choice_d	brown
choice_e	black
choice_f	white
choice_g	
answer_a	Incorrect
answer_b	Correct

Bei `question` und `choice_a` bis `choise_g` handelt es sich um Variablen, bei denen man Werte bzw. Texte eingeben kann. Bei `answer_a` bis `answer_h` handelt es sich um den Typ `List`, der nur verschiedene Werte zulässt: In diesem Fall nur `Correct` oder `Incorrect`. Diese Variablen teilen dem Smartclip mit, welche Antworten richtig und welche falsch sind.

Ordner Smartdi, Datei: grundSC.fla

Nach dem Abändern der Werte sieht der Smartclip so aus:



Die Werte und Variablen sind natürlich von Smartclip zu Smartclip unterschiedlich und sind vom jeweiligen Entwickler individuell erstellt worden. Möglich wären hier auch noch Objekttypen, die wieder verschiedene Variablen enthalten können.

Möglich ist auch, eine benutzerdefinierte Oberfläche zu dem Smartclip zu erstellen. Diese ist dann ein SWF, das es ermöglicht, durch Verschieben, Klicken oder Eingabe von Werten das Verhalten des Smartclips zu verändern.

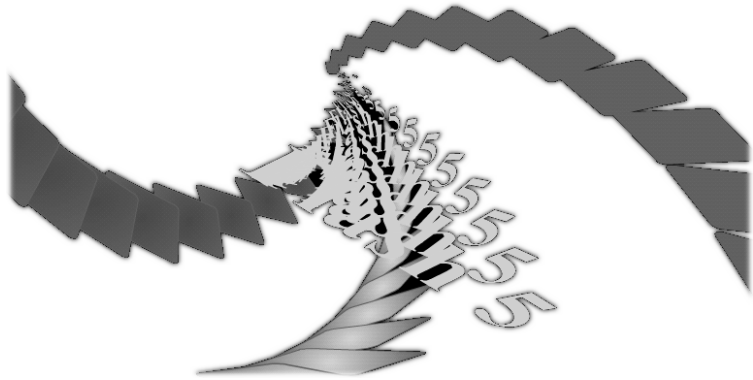
Benutzerdefinierte Oberfläche

→ *92 Erstellen von Smartclips – Benutzerdefinierte Oberfläche.*

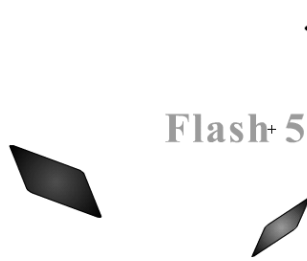
5.2 Effekte

Ordner Texteffe, Datei: texteffe.fla

Timeslide (Texteffekt)

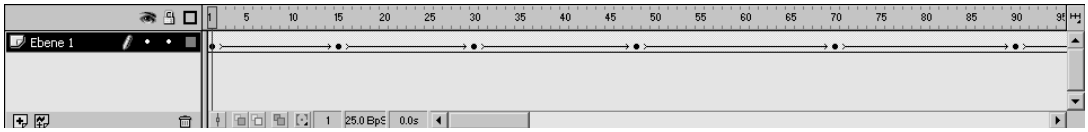


Anhand des folgenden Beispiels zeigen wir Ihnen, wie Sie mit einfachen Mitteln einen eindrucksvollen und doch wenig speicherintensiven Texteffekt erzeugen können. Hierfür werden der Tweeningeffekt, verschiedene Farbeinstellungen und der `duplicateMovieClip`-Befehl benutzt.

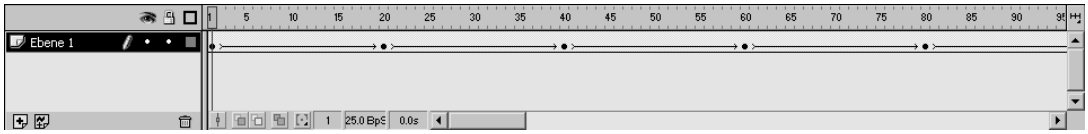


Der ganze Effekt besteht aus einer Grafik. Diese wird mit Hilfe des Tweeningeffektes bewegt. Damit dies nicht direkt auf der Hauptzeitleiste passiert und weil die Grafik dupliziert werden muss, baut man diese mit dem Bewegungstweening in ein Movie ein. Das Movie hat den Instanznamen *Grafik*.

In diesem Movie befindet sich ein weiteres Movie, da auf die Grafik noch ein Farbeffekt angewendet wird.

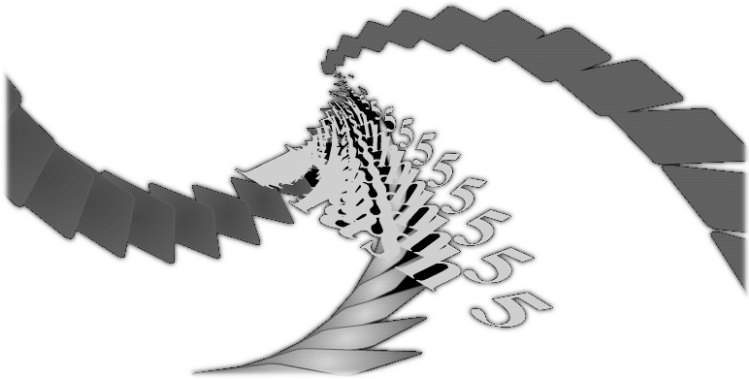


Im zweiten Movie befindet sich ein weiteres Tweening, bei dem die Farbwerte der Grafik verändert werden.



Man erkennt sofort, dass die Punkte des Farbwechsels und des Richtungswechsels asynchron verlaufen; die zwei Movies haben auch unterschiedliche Längen. Dadurch erhält man den Effekt, dass bei gleichartigen Bewegungen nicht immer die selben Farbverläufe entstehen.

Sie können dem Ganzen nun Tiefe bzw. einen dreidimensionalen Charakter verleihen, indem Sie das Movie *Grafik* mit dem `duplicateMovieClip`-Befehl mehrmals duplizieren und die duplizierten Movies ein wenig umpositionieren, drehen, verkleinern oder vergrößern.



Mit diesem Effekt lassen sich mit geringen Mitteln und kleinen Dateigrößen beeindruckende Movies erzeugen. Man kann ihn auch beim Preloader einsetzen, um den Ladestatus anzuzeigen, indem man keinen statischen Text sondern einen dynamischen Text benutzt, der z. B. immer die aktuelle Prozentzahl angibt.

Das ActionScript mit dem `duplicateMovieClip`-Befehl, das im ersten Frame auf der Hauptbühne liegt, sieht wie folgt aus:

```
for (i=1; i<15; i++) {
    duplicateMovieClip ("_root.grafik", "grafik"+i, i);
    _root["grafik"+i]._yscale   =i*i;
    _root["grafik"+i]._xscale   =i*12;
    _root["grafik"+i]._rotation =i*8;
    _root["grafik"+i]._y        =i*10+150;
}
_root.grafik.gotoAndStop(1);
stop ();
```

Hier werden 14 Kopien des Movies *Grafik* angefertigt und deren Position variabel angepasst, je nachdem, um die wievielte Kopie es sich handelt. Die späteren Kopien werden auf dem Bildschirm immer größer dargestellt, da i sich erhöht und dadurch die errechneten Werte für den `_xscale` oder `_yscale` auch vergrößern. Leider beansprucht die Darstellung des Bewegungsablaufs mit mehreren duplizierten Movies sehr viel Rechnerleistung, so dass man rasch an die Grenze des Machbaren gelangt und das SWF beim

Abspielen unter Umständen ruckelt. In der vorletzten Zeile wird das SWF, das außerhalb der Bühne liegt gestoppt, damit dieses nicht aus Versehen im Hauptbild auftaucht.

Texteffekt.fla

▶ **Texteinblendung**

Um eine schlichte Texteinblendung zu verwirklichen, verwenden einen Maskeneffekt. Legen Sie zunächst eine Filmsequenz an, in der Ihr Text in einem statischen Textfeld als Maske angelegt ist.

In der maskierte Ebene erstellen Sie ein Bewegungstweening mit einem Verlauf (Hintergrundfarbe zu Textfarbe) mit anschließender Fläche in der gewünschten Textfarbe.

Das Ergebnis sieht wie folgt aus:

Wenn Sie diese nun abspielen, sieht der Effekt wie folgt aus:

Mauseffekte/Maustrailer

Maustrailer sind Effekte, die in Abhängigkeit zum Mauszeiger stehen. Dies kann ein Fadenkreuz, ein rotierender Stern oder auch die Ausgabe von Mauskoordinaten sein.

▶ **Mauskoordinaten direkt am Mauszeiger ausgeben (Flash 4)**

Dieser Effekt veranschaulicht das Zuordnen von Variablen sowie das Zusammensetzen von String-Ausdrücken.

Zuerst erstellen Sie auf der Hauptbühne einen Movieclip mit dem Instanznamen „trail“. Dieser Name ist beliebig, wird aber im Folgenden verwendet. Der erste Frame auf der Hauptbühne enthält folgendes ActionScript:

```
Start Drag ("/trail", lockcenter)
```

Hier wird der soeben erstellte Movieclip an die aktuellen Mauskoordinaten geheftet. Der Movieclip „trail“ selbst enthält ein Textfeld, das den Variablennamen „ausgabe“ erhält. Im ersten Frame des Movieclips werden die Mauskoordinaten mit Hilfe des Movieclips ausgelesen, in einen String gesetzt und dem Textfeld zugeordnet:

```
Set Variable: "X" = GetProperty ("",_x)
Set Variable: "Y" = GetProperty ("",_y)
Set Variable: "ausgabe" = X&", "&Y
```

Mit Hilfe von & werden Variablen und Textausdrücke verkettet. Damit die Koordinatenangaben ständig aktualisiert werden, muss im darauf folgenden Frame die folgende Aktion eingetragen werden:

```
Go to and Play (1)
```

Dieser Befehl bewirkt, dass eine Endlos-Schleife entsteht und somit die Werte laufend aktualisiert werden.

▶ Alternative Mauszeiger erzeugen (Flash 5)

Flash 5 verfügt über die Möglichkeit, den System-Mauszeiger über dem aktivem Flashfilm auszublenden. Der entsprechende ActionScript-Befehl lautet:

```
Mouse.hide();
```

Um dem Benutzer weiterhin die Möglichkeit der Navigation zu bieten, muss an Stelle des Mauszeigers ein anderes Symbol eingesetzt werden. (Ausnahme sind z. B. Spiele mit Tastatursteuerung.)

Mit Hilfe des `onClipEvent (Ereignis)` frägt der Film, der den Mauszeiger ersetzen soll, automatisch laufend ab, ob

- der Cursor bewegt wurde (`mouseMove`) – wenn ja, wird der alternative Mauszeiger nachgeführt.
- eine Maustaste gedrückt (`mouseDown`) wird – wenn ja, nimmt der Cursor das Erscheinungsbild an, das im Frame mit dem Namen „pressed“ hinterlegt wurde.
- eine Maustaste losgelassen (`mouseUp`) wird – wenn ja, nimmt der Cursor das Erscheinungsbild an, das im Frame mit dem Namen „released“ hinterlegt wurde.

Beim Laden des Cursors (`load`) bleibt der System-Cursor verborgen. Die Anweisung `stop()` bewirkt, dass nicht alle möglichen Cursor, die in der gleichen Filminstanz liegen, durchlaufen werden.

```
onClipEvent (load) {
    Mouse.hide();
    stop();
}
```

Sobald die Maus bewegt wird (`mouseMove`), werden die X-Y-Koordinaten auf die Instanz angewandt. Mit `_x` werden die instanzeigenen Koordinaten mit denen des Mauscursors (unsichtbar) gleichgesetzt (`_root._xmouse`). Da wir davon ausgehen, dass sich der Mauszeiger nachher an der Hauptbühne ausrichten soll, legt `_root` den Bezugspunkt fest, der auch für die Bestimmung von`_xmouse` herangezogen werden muss. Gleiches gilt für die Y-Koordinate.

```
onClipEvent (mouseMove) {
    _x = _root._xmouse;
    _y = _root._ymouse;
    updateAfterEvent();
}
```

Ist eine Maustaste gedrückt, springt die Instanz in den Frame „pressed“, wo ein entsprechendes Bild eingefügt ist. Dort bleibt der Film stehen, so lange nicht einer der anderen Events (`mouseUp`) eintritt.

```
onClipEvent (mouseDown) {
    this.gotoAndStop("pressed")
    updateAfterEvent();
}
```

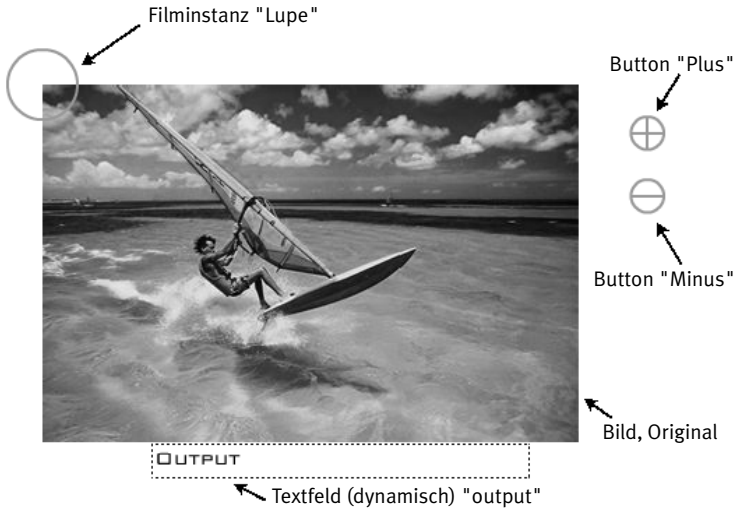
Wird die Maustaste losgelassen, springt die Instanz in den Frame „released“, wo ein entsprechendes Bild (normalerweise das Erscheinungsbild des „Ausgangscursors“) eingefügt ist. Dort bleibt der Film stehen, so lange nicht einer der anderen Events (`mouseDown`) eintritt.

```
onClipEvent (mouseUp) {
    this.gotoAndStop("released");
    updateAfterEvent();
}
```

Maskeneffekte (Lupe)

Lupe.fla

Eine Frage, die immer wieder gestellt wird, ist, wie mit Flash eine Lupe zu realisieren ist. Auf den ersten Blick ist dies wenig problematisch, doch sehr bald stellen Sie fest, dass es nicht möglich ist, per `startDrag` eine Maske über den maskierten Bereich zu ziehen. An dieser Stelle kann man aber zu einem Trick greifen: Legen Sie eine Instanz für die Lupe an und kopieren Sie hier Ihr Originalbild hinein. Dieses ist unter der Maske per ActionScript ansprechbar und „beweglich“. Wenn Sie nun Original und Vergrößerung aufeinander abstimmen, erhalten Sie Ihren Lupeneffekt. Die folgende Abbildung zeigt die Anordnung aller Objekte auf der Hauptbühne:



Hinter den Buttons verbergen sich nur kurze Scripts, um die Variable *faktor* zu verändern.

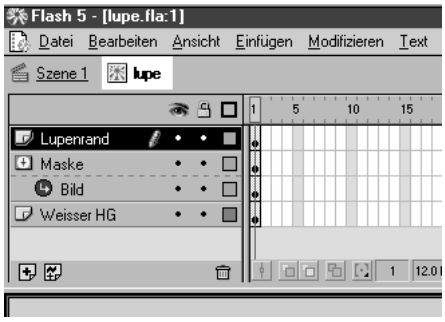
Button Minus:

```
on (press){
_level0.lupe.faktor+=10;
}
```

Button Plus:

```
on (press){
_level0.lupe.faktor-=10;
}
```

Die aktuellen `faktor`-Werte werden später immer in der `output`-Anzeige angegeben. Die Lupe selbst enthält vier Ebenen, die wir im Folgenden erläutern:



Die oberste Ebene *Lupenrand* beinhaltet ausschließlich den Rahmen der Lupe, oder vielmehr die Fassung. Die zweite Ebene *Maske* enthält eine Maske,die genau passend zum Rand sein sollte. Die dritte Ebene enthält die Instanz *Bild* und das Bild surfer.jpg in Originalgröße; die vierte und unterste Ebene enthält eine Kopie der Maske in der Farbe des Hintergrundes, um am Rand des Bildes dieses nicht doppelt zu sehen. Folgendes ActionScript steuert nun die Aktionen:

```
onClipEvent (load) {
      startDrag ("", true);
      var faktor = 200;
```

Die Lupe wird an den Cursor geheftet, der Zoom auf 200% voreingestellt.

```
}
onClipEvent (enterFrame) {
      _level0.lupe.bild._x = (_level0.lupe.faktor/
100)*_level0.ursprung._x-(_level0.lupe.faktor/
100)*(_root._xmouse);
      _level0.lupe.bild._y = (_level0.lupe.faktor/
100)*_level0.ursprung._y-(_level0.lupe.faktor/
100)*(_root._ymouse);
      _level0.lupe.bild._xscale = (_level0.lupe.faktor);
      _level0.lupe.bild._yscale = (_level0.lupe.faktor);
```

Die Koordinaten der Bilder (_levelo.ursprung und _levelo.lupe.bild) werden abgestimmt, die Vergrößerung gesetzt.

```
      _level0.output = _level0.lupe.faktor+"% des Orginals";
```

Der aktuelle Zoomfaktor wird ausgegeben.

```
}
```

Dieses Script liegt in der Instanz *Lupe* auf der Hauptbühne.

Soundeffekte

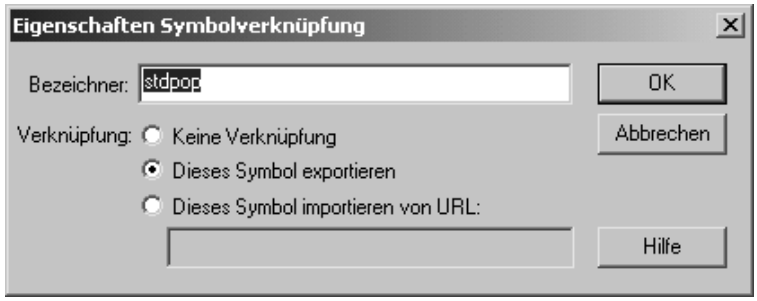
Ordner Soundeff, Dateien: soundeff.fla und flugzeug.fla

In Flash 5 ist es möglich, mit ActionScript die Volume(Lautstärke) und Pan(Balance)-Einstellungen zu ändern. Dies geschieht mit einem Soundobjekt.

Nachdem die wichtigsten Grundkenntnisse im Abschnitt *Sound* → 62 schon erläutert wurden, kommen wir nun direkt auf das Wesentliche des Soundobjektes. Die erste Frage, die Sie sich vielleicht stellen könnten, wäre, wozu das Soundobjekt überhaupt benötigt wird. Die zwei häufigsten Anwendungen sind einerseits die Erstellung eines Pan(Balance)-Reglers bzw. eines Volume(Lautstärke)-Reglers, andererseits die Unterlegung von User-gesteuerten Grafiken mit Soundeffekten, bei denen sich Balance und Lautstärke anpassen.

Um das Soundobjekt überhaupt benutzen zu können, müssen Sie das entsprechende Sample vorab diesem Soundobjekt zuweisen. Hierfür müssen Sie dem Sample einen Verknüpfungsnamen geben.

- Importieren Sie ein Sample in Flash (→ 62 *Sound*).
- Öffnen Sie die Bibliothek Ihres Filmes (STRG + L).
- Klicken Sie mit der rechten Maustaste auf das Sample in der Bibliothek Ihres Filmes und wählen Sie anschließend die Option VERKNÜPFUNG .



- Wählen den Menüpunkt VERKNÜPFUNG › DIESES SYMBOL EXPORTIEREN aus.
- Bei Bezeichner geben Sie den Namen, den die Verknüpfung haben soll, ein.
- Bestätigen Sie mit „OK“.

In diesem Beispiel wurde dadurch dem Sample der Verknüpfungsname `stdpop` zu gewiesen.

On ClipEvent
→*Kapitel 2: Neuerungen in Flash 5.*

Um den Sound zu starten, erstellen Sie zunächst ein Soundobjekt. Dieses Soundobjekt wird wie alle anderen Objekte auch mit `new name` erstellt. Da das Objekt am besten direkt zu Beginn hinzugefügt wird, macht man dies am besten über `onClipevent(load)`. Da Clipevents aber leider nur auf Movies angewendet werden können, sollte man sich ein Movie in dem Film aussuchen, in dem man alle Clipevents einbaut, so dass man die Befehle auch wieder findet. Es spricht allerdings nichts dagegen, einen Befehl in ein anderes Movie einzubauen. Allerdings sollten alle `onClipevent(load)`, in denen eventuel noch Funktionen definiert werden, zusammen in einem Movieclip liegen.

```
onClipEvent (load) {
    standardpop = new Sound();
    standardpop.attachSound("stdpop");
}
```

`onClipevent (load)` besagt, dass der folgende Befehl einmal ausgeführt wird – und zwar dann, wenn der Movieclip direkt zu Beginn eingeladen wird.

Der erste Befehl erzeugt ein Soundobjekt mit dem Namen „standardpop“. Der zweite Befehl übergibt das Sample mit dem Verknüpfungsnamen `stdpop` an das Soundobjekt. Um nun das Musiksample zu starten oder zu stoppen, sprechen Sie direkt das Soundobjekt an.

```
on (release) {
standardpop.stop()
}
```

Stoppen oder Starten des Sounds:

```
on (release) {
    standardpop.stop()
    standardpop.start(0,20);
}
```

Der Stoppbefehl steht dort nur, damit das Sample nicht doppelt abgespielt wird, wenn der Besucher zweimal auf Start drückt. Das Sample wird stattdessen gestoppt und anschließend neu gestartet. Die Parameter, die mitübergeben werden, sind die Startposition und die Anzahl der Wiederholungen. Dieses Sample würde direkt am Anfang des Samples starten und 20mal wiederholt werden.

Das Soundobjekt bietet aber noch eine Vielzahl weiterer Möglichkeiten: Zum Beispiel kann man die Eigenschaften Lautstärke (Volumen) und Balance (Pan) auslesen. Dies geschieht auch in der Beispielsdatei bei den beiden Textfeldern. Da man für die Textfelder wieder ein `onClipEvent` braucht, erstellt man beide Textfelder direkt in einem Movieclip. Theoretisch hätte man diese Befehle auch in den ersten Movieclip einbauen können, man hätte dann aber absolute Pfadangaben machen müssen. Außerdem wird die Gliederung auf diese Weise übersichtlicher. Dies ist aber natürlich nicht zwingend notwendig. Wir haben die Befehle in einem eigenen Movieclip erstellt, um die Übersicht zu behalten. Diese Vorgehensweise ist aber nicht zwingend. Durch das Erstellen eines einzelnen Movies nur für die Textboxen wird dies aber sofort ersichtlich.

```
onClipEvent (enterFrame) {
    vol = _root.standardpop.getVolume();
    pan = _root.standardpop.getPan();
}
```

`OnClipEvent (enterFrame);` Diese Aktion sorgt dafür, dass die enthaltenen Befehle immer wieder abgearbeitet werden. In diesem Fall werden den Variablen `vol` und `pan` immer wieder die Ergebnisse der Funktionen `getPan` und `getVolume` zugewiesen. GetPan gibt die Einstellungen der Balance wieder. Dieser Wert kann zwischen -100 und 100 liegen und wird als ganze Zahl ausgegeben. `GetVolume` gibt die Einstellung der Lautstärke wieder. Dieser Wert liegt zwischen 0 und 100 und wird als ganze Zahl ausgegeben. Da die Textfelder die Zuweisung zu den Variablen `vol` und `pan` haben, zeigen sie immer die aktuellen Einstellungen der Lautstärke und der Balance an.

Den interessanteren Teil des Soundobjekts dürfte der Zugriff auf die Einstellungen des Sounds über das ActionScript darstellen. In der Beispielsdatei haben wir zwei Regler erstellt, über die man die Lautstärke wie auch die Balance ändern kann.

Wir erklären zunächst das Funktionsprinzip eines solchen Reglers. Der erstellte Regler ist vom ActionScript her sehr universell einsetzbar, da man das ActionScript nicht ändern braucht, um die Länge zu ändern. Zuerst er-

kennt man wieder ein Movieclip, um das Ganze zusammenzuhalten, damit die einzelnen Objekte nicht direkt auf der Hauptbühne liegen. Theoretisch könnte man dieses Movieclip auch als Smartmovieclip der Bibliothek hinzufügen. In diesem Movie befinden sich zwei weitere Movies: eines mit dem Instanznamen *Länge*, das die Strecke des Reglers enthält, zum anderen der Regler selbst, der den Instanznamen *Regler2* besitzt. Will man einen Regler mit einem längeren Weg erzeugen, muss man so beim Erstellen lediglich mit der Maus die Länge des Movies verändern. Würde es sich nicht um ein Movieclip handeln, wäre dies so nicht möglich. Der Regler liegt in einem Movie, damit dieser mit der Aktion startDrag an den Mauszeiger „geheftet" werden kann.

Der Regler ist als Button ausgeführt und besitzt folgendes ActionScript:

```
on (press) {
    startDrag (this, false, _parent.laenge._x -
_parent.laenge._width/2, _parent.laenge._y, _parent.laenge._x +
_parent.laenge._width/2, _parent.laenge._y);
}
on (release, releaseOutside) {
    stopDrag ();
}
```

Beim Drücken des Schalters wird also `startDrag` ausgeführt.

Die Aktion `startDrag` besitzt folgende Argumente:

Allgemein ausformuliert: `StartDrag` (Instanzname,Objekt unter Mauszeiger zentrieren, true/false, max. linker X-Wert für das Objekt, max. oberer Y-Wert für das Objekt, max. linker X-Wert für das Objekt, max. unterer Y-Wert für das Objekt);

Die `startDrag`-Aktion wird auf das Movie „this" angewandt, `This` ist ein Synonym für, das Movie, in dem sich der Befehlsaufruf befindet. Der kleinste X-Wert entspricht dem `_parent.laenge._x - _parent.laenge._width/2`, also der X-Position des Movieclips *Länge* weniger der halben Breite des Movieclips *Länge*. Dadurch erhält man den linken äußeren Punkt, bis zu dem der Regler maximal gezogen werden darf. Der rechte Rand errechnet sich genau auf die gleiche Weise, nur dass man die Hälfte nicht abzieht sondern dazu addiert. Da der Regler nur waagerecht verschoben werden soll, wird die Y-Position für den Regler auf einen konstanten Wert gesetzt. Damit der Regler genau mittig erscheint, wird dieser auf den Y-Wert des Movieclips *Länge* gesetzt. Wenn man nun den Movieclip *Länge* verändert, verändert man auch die Reichweite des Reglers.

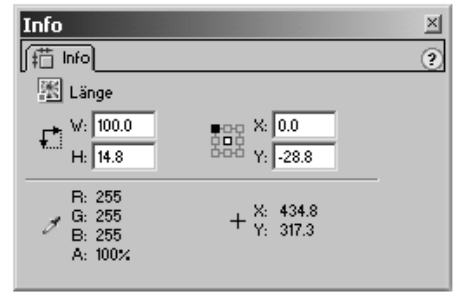
Anstatt den Rand des Movies über die Position und die Breite zu ermitteln, kann man auch das getBounds-Objekt verwenden.

Die Aktion `stopDrag` wird beim Loslassen der Maustaste ausgeführt und bewirkt, dass der Movieclip *Regler* nicht mehr dem Mausezeiger folgt.

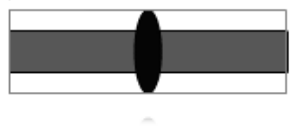
Dadurch hat man nun einen beweglichen Regler. Dieser hat aber noch keine Funktion. Um die Lautstärke analog zu der Reglerstellung zu ändern, muss man fortlaufend die Stellung des Reglers überprüfen und die Stellung der Lautstärke zuweisen.

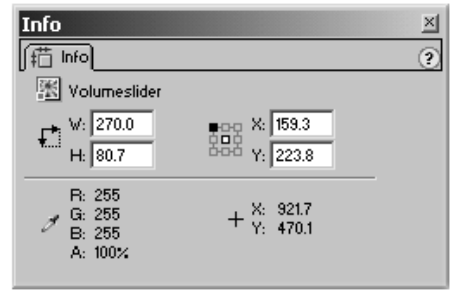
```
onClipEvent (enterFrame) {
    _root.standardpop.setVolume(Regler2._x);
}
```

Durch diesen Befehl entspricht die Lautstärke des Sounds *standardpop* (Verknüpfungsname) stets der Reglerstellung. Da die Lautstärke zwischen 0 und 100 liegen sollte, muss man nun noch dafür sorgen, dass die Minimalstellung des Reglers x = 0 beträgt und die Maximalstellung x=100. Wenn man die Breite bzw. die Länge des Movieclips *Länge* genau auf 100 setzt und die x-Position auf 0, entspricht dies genau den gewünschten Parametern. Ändert man nun im übrigen auf der Hauptbühne die Größe des Movies *Volume*, hat dies keinen Einfluss auf die Größe der darin liegenden Movies. Da der Movieclip *Länge* in dem Movieclip *Volume* sitzt, der auf der Hauptbühne verbreitert wurde, scheint es so, als ob die Reglerbreite sich ebenfalls vergrößert hätte. Dies ist aber nur optisch der Fall, die X-Y Koordinaten in den Movieclips bleiben erhalten, so dass der Movieclip *Länge* weiterhin eine Breite von 100 besitzt.



In dem Movieclip `Regler` besitzt der Movieclip L*änge* genau eine Breite von 100 und genau auf dieser Grundlage wird die X-Position des Reglers abgefragt.

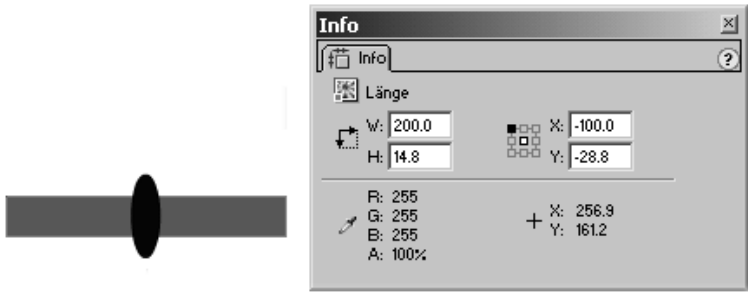




Auf der Hauptbühne wurde der Movie-Regler anschließend gestreckt, damit der Regler groß genug auf dem Bildschirm erscheint. Dies hat aber keinen Einfluss auf die darin liegende Abfrage und die X-Werte.

Für den Balance- bzw. Panregler gelten fast genau dieselben Einstellungen. Einzig und alleine der Movieclip *Länge* in dem Movieclip *Regler* wird

abgeändert, da der Panregler nicht Werte von 0 bis 100 erzeugen soll, sondern von -100 bis 100. Wichtig dabei ist die X-Position und die Breite. Die Y-Position sowie die Höhe sind irrelevant, da dem Regler ohnehin eine feste Y-Position zugewiesen wird. Deshalb sehen die Einstellungen wie folgt aus:



Nun, da die Lautstärke und die Balance jeweils automatisch an die Position des Reglers angepasst werden, kann man diese Werte per ActionScript noch automatisch verändern lassen. Dies ist sehr nützlich, wenn man auf einer Homepage den Sound bzw. die Musik ein- oder ausblenden möchte. Da wir in diesem Beispiel Regler benutzt haben, setzen wir immer direkt die X-Werte der Regler ein, um die Lautstärke- oder Paneinstellungen zu ändern. Wenn Sie die Regler nicht benutzen, müssen Sie die entsprechende Zeile im ActionScript, in dem die X-Position des Reglers angegeben wird, gegen eine entsprechende Soundvolumenangabe tauschen.

Der Schalter zum Einblenden eines Samples befindet sich in einem Movieclip, welches direkt im ersten Frame stoppt. Wird der Schalter gedrückt, so springt der Movieclip in den zweiten Frame. In diesem wird dann folgendes ActionScript ausgeführt.

```
vol=_root.Reglervol.Regler2._x;
_root.fadeout.gotoAndStop ( 1 );
_root.special.gotoAndStop ( 1 );
```

Zuerst wird in der Variable `vol` die X-Position des Lautstärkereglers gespeichert. Diese Position entspricht der augenblicklichen Lautstärke.

Die zwei folgenden `gotoAndStop (1);`-Anweisungen sorgen dafür, dass diese Aktion abgebrochen wird, falls zuvor die Ausblende- oder die Special-Taste gedrückt wurde.

Im dritten Frame wird zuerst geprüft, ob die maximale Lautstärke von 100 bereits erreicht ist. Falls dem nicht so ist, wird die Lautstärke um 1 erhöht und zum nächsten Frame gesprungen, welcher wieder auf den zweiten Frame verweist. Das Ganze wiederholt sich solange, bis die maximale Lautstärke von 100 erreicht wurde; dann wird der Abspielvorgang des Einblende-Movies gestoppt.

```
if (vol>=100) {
    gotoAndStop (1);
} else {
    vol = _root.Reglervol.Regler2._x;
    vol++;
    _root.Reglervol.Regler2._x = vol;
}
```

Die Befehlszeile `vol = _root.Reglervol.Regler2._x;` steht dort, damit der User auch während des Ausblendvorgangs die Möglichkeit hat, mit der Maus die Reglerposition zu ändern.

Ohne diese Zeile würde der Regler kontinuierlich auf 100 zusteuern. So wird zwischendurch noch einmal abgefragt, ob der Regler seine Position geändert hat und falls dies zutrifft, wird die Variable `vol` diesem neuen Wert angepasst.

Nebenbei bemerkt kann man all diese Befehle genauso gut in einem `onClipEvent()` erstellen.

Der Button für das Ausblenden ist nach genau demselbem Schema aufgebaut. Der einzige Unterschied: Die Variable `vol` wird nicht aufsteigend, sondern absteigend gezählt. Entsprechend wird das Abspielen des Movies bei 0 angehalten und nicht bei 100.

```
if (vol<=0) {
    gotoAndStop (1);
} else {
    vol = _root.Reglervol.Regler2._x;
    vol--;
    _root.Reglervol.Regler2._x = vol;
}
```

Der Special-Schalter basiert ebenfalls auf dem gleichen Prinzip, da sich fast alles weitere mit diesem Aufbau erzeugen lässt. Es wurden nur zwei weitere Variablen eingeführt, um die Laufrichtung der Regler zu bestimmen – `VolUp` und `PanUp`.

```
if (vol<20) {
volup=true;
} else if (vol>100) {
volup=false;
}
if (volup==true) {
vol++;
} else {
vol--;
}
_root.slidervol.slider2._x=vol;
```

```
if (pan<-80) {
panup=true;
} else if (pan>80) {
panup=false;
}
if (panup==true) {
pan++;
} else {
pan--;
}
_root.sliderpan.slider2._x=pan;
```

Da in diesem Frame nicht die Position der Regler neu abgefragt wird, ist ein Eingreifen des Users während des Ablaufens des Effektes nicht möglich.

Auf der CD befindet sich auch noch eine zweite Datei mit einem Beispiel für das Soundobjekt.

Das Soundobjekt besitzt noch die Methode `setTransform`. Mit dieser Methode ist es möglich, auf die Ausgabekanäle des Sounds zuzugreifen: z.B. besitzt ein Stereo-Sample zwei Kanäle (rechts und links). Normalerweise wird das Sample so abgespielt, dass der rechte Kanal des Samples auf dem rechten Ausgang liegt und der linke Kanal auf dem linken Ausgang.

Der Aufruf des `SoundTransform`-Objektes sieht wie folgt aus.

```
on (release) {
TransformObjektname = new Objekt();
TransformObjektname= { ll: '100', lr: '0', rr: '100', rl: '0'};
Sound.setTransform(TransformObjektname);
Sound.start(0,1);
}
```

Erstellen Sie zuerst ein Objekt. Dies geschieht in der ersten Zeile. Das Objekt hat hier den Namen *TransformObjektname*. In der zweiten Zeile werden dem Objekt Werte zugewiesen.

LL	steht für den linken Kanal des Samples; das zweite L gibt an, dass dieser auf dem linken Ausgang wiedergegeben wird.
LL	bedeutet linker Kanal auf dem linken Ausgang (Standard = 100).
LR	bedeutet linker Kanal auf dem rechten Ausgang (Standard = 0).
RR	bedeutet rechter Kanal auf dem rechten Ausgang (Standard = 100).
RL	bedeutet rechter Kanal auf dem linken Ausgang (Standard = 0).

Die Zuordnung funktioniert prozentweise. Man sollte darauf achten, einem Ausgang nie über 100 zuzuweisen, da es sonst zu einer leichten Übersteuerung kommt.

Im Normalfall wird also der rechte Kanal dem rechten Ausgang zu geordnet und der linke Kanal dem linken Ausgang. Diese Einstellungen kann man nun verändern. Dadurch kann man bei einem Stereosampel den Stereoeffekt umkehren. Dafür würden die Einstellungen wie folgt aussehen:

```
TransformObjektname= { ll: '0', lr: '100', rr: '0', rl: '100'};
```

Bei diesem Befehl werden die Kanäle gewechselt. Möglich wäre auch den Stereoeffekt „auszuschalten“:

```
TransformObjektname= { ll: '50', lr: '50', rr: '50', rl: '50'};
```

Man kann aus einem Stereosound auch einen Monosound machen.

```
TransformObjektname= { ll: '100', lr: '100', rr: '0', rl: '0'};
```

Hierbei würde jetzt der linke Kanal auf beiden Ausgängen wieder gegeben, der rechte Kanal wird nicht ausgegeben.

Erst durch die Zuordnung des Objektes *TransformObjektname* zu der Methode `setTransform` und dem davor zu setzenden Soundobjekt wird dieser Befehl aktiv.

```
Sound.setTransform(TransformObjektname);
```

Dieses Objekt funktioniert also wiederum nur mit dem Soundobjekt zusammen. Die Methoden `setPan` und `setVolume` zu `setTransform` überschreiben sich gegenseitig.

Hat man z. B. die Lautstärke auf 70 und die Balance auf 50 gesetzt und wendet nun `setTransform` an, so werden die Werte von `setPan` und `setVolume` ungültig und es gelten die neuen Einstellungen von `setTransform`. Dies gilt natürlich auch umgekehrt.

5.3 Erweiterte Funktionen

Externe Dateien einlesen

Es ist möglich, in Flash „externe“ Daten zu importieren. Diese Daten müssen allerdings nach einer bestimmten Syntax aufgebaut sein. Des Weiteren erläutern wir das Einlesen von HTML-Dateien.

▶ **Textdateien**

Durch den Befehl `loadVariablesNum("http://Server.de/Verzeichnisse/datei.txt", 0);` lesen Sie Daten aus einer Datei ein. Der Dateinamen oder die Dateiendung ist dabei unwichtig. Sie muss lediglich einer gewissen Syntax entsprechen und sich auf demselben Server bzw. in derselben Subdomain befinden. Flash setzt die entsprechenden Variablen durch das Einlesen der Datei in den entsprechenden Level oder Movie. o bedeutet, dass die Variablen in `_Level0 (_root)` eingeladen werden. Sie können die Variablen aber auch direkt in einen Film einladen, um so eine bessere Übersicht über die Variablen zu erhalten.

```
loadVariables ("http://www.Server.de/Name.txt", "instanzname");
```

- **Data.txt**

```
inhalt=Das hier ist der Text aus der Textdatei. Mit 2 Sonderzeichen : %26, %25
&inhalt2=Das hier ist der 2´te Text aus der Textdatei mit ä,ü und ö
&Variable=1&
&Variable2=1&
```

In Flash 4 war es möglich, Variablen durch zwei &-Zeichen, zu Beginn und am Ende, zu deklarieren und mit einem &-Zeichen, Strings, die keine Sonderzeichen enthielten, an Flash zu übermitteln.

In Flash 5 hat sich dies geändert. Wenn man eine TXT-Datei erstellt, die nur ein &-Zeichen pro zu übergebende Variable enthält, kann man diese Variable unter Flash nur lesen bzw. ausgeben. Mit dieser Variable können Sie nicht direkt Berechnungen oder Stringoperationen erstellen. Dazu müsste man diese erst umwandeln, z. B.: mit `parseFloat()`. Deshalb ist es bei Flash 5 auch sinnvoller, alle Variablen in der TXT.-Datei direkt mit zwei &-Zeichen zu versehen. Dadurch wird diese direkt als String übergeben. Wenn man mit diesem Wert allerdings rechnen möchte, muss man ihn trotzdem mit `parseFloat()` in eine Zahl umwandeln.

Wenn Sie Sonderzeichen darstellen möchten, zum Beispiel ein &-Zeichen, welches für die Verkettung der Variabeln reserviert ist, gibt es einen entsprechenden Code, bei dem Sie das jeweilige Zeichen durch das entsprechende Codezeichen ersetzen (wie oben in der TXT-Datei schon angedeutet).

- **URL-Encoding-Tabelle**

Zeichen	Code	Zeichen	Code	Zeichen	Code
backspace	%08	[	%5B	Ê	%CA
tab	%09	\	%5C	Ë	%CB
space	+ or %20	]	%5D	Ì	%CC
newline	%0A	^	%5E	Í	%CD
Carriage return	%0D	_	%5F	Î	%CE
!	%21	`	%60	Ï	%CF
"	%22	a	%61	ð	%D0
#	%23	b	%62	Ñ	%D1
$	%24	c	%63	Ò	%D2
%	%25	d	%64	Ó	%D3
&	%26	e	%65	Ô	%D4
'	%27	f	%66	Õ	%D5
(	%28	g	%67	Ö	%D6
)	%29	h	%68	Ø	%D8

Die Tabulatortaste (%09) wird von Flash und HTML nicht unterstützt, da diese Taste schon mit dem Wechsel des Focus zwischen den Textfeldern belegt ist.

Zeichen	Code	Zeichen	Code	Zeichen	Code
*	%2°	i	%69	Ù	%D9
+	%2B	j	%6A	Ú	%DA
,	%2C	k	%6B	Û	%DB
-	%2D	l	%6C	Ü	%DC
.	%2E	m	%6D	∫	%DD
/	%2F	n	%6E	Ω	%DE
0	%30	o	%6F	ß	%DF
1	%31	p	%70	à	%E0
2	%32	q	%71	á	%E1
3	%33	r	%72	â	%E2
4	%34	s	%73	ã	%E3
5	%35	t	%74	ä	%E4
6	%36	u	%75	å	%E5
7	%37	v	%76	æ	%E6
8	%38	w	%77	ç	%E7
9	%39	x	%78	è	%E8
:	%3A	y	%79	é	%E9
;	%3B	z	%7A	ê	%EA
<	%3C	{	%7B	ë	%EB
=	%3D	|	%7C	ì	%EC
>	%3E	}	%7D	í	%ED
?	%3F	~	%7E	î	%EE
@	%40	¢	%A2	ï	%EF
A	%41	£	%A3	Π	%F0
B	%42	¥	%A5	ñ	%F1
C	%43	|	%A6	ò	%F2
D	%44	§	%A7	ó	%F3
E	%45	«	%AB	ô	%F4
F	%46	¬	%AC	õ	%F5
G	%47	¯	%AD	ö	%F6
H	%48	°	%B0	ffi	%F7
I	%49	±	%B1	ø	%F8
J	%4A	ª	%B2	ù	%F9
K	%4B	,	%B4	ú	%FA
L	%4C	µ	%B5	û	%FB
M	%4D	»	%BB	ü	%FC
N	%4E	≠	%BC	fi	%FD
O	%4F	∞	%BD	fl	%FE
P	%50	¿	%BF	ÿ	%FF

Zeichen	Code	Zeichen	Code	Zeichen	Code
Q	%51	À	%C0		
R	%52	Á	%C1		
S	%53	Â	%C2		
T	%54	Ã	%C3		
U	%55	Ä	%C4		
V	%56	Å	%C5		
W	%57	Æ	%C6		
X	%58	Ç	%C7		
Y	%59	È	%C8		
Z	%5A	É	%C9		

Wenn Sie eine TXT-Datei öfters von einem Server herunterladen müssen, weil sich deren Inhalt geändert hat, ensteht das Problem, dass der Browser diese Datei aus dem Cache liest. Um dies zu vermeiden, besteht die Möglichkeit, dem Aufruf der Datei unter Flash ein „?“ anzuhängen. Dadurch teilt man dem Browser mit, dass man Variablen mit an die Datei übergeben möchte. Dies unterlassen wir zwar, aber der Browser liest dadurch die TXT-Datei nicht mehr aus dem Cache, sondern lädt sie neu herunter:

→ 110 CGI / Schreiben und Lesen von Textdateien.

```
loadVariables ("http://www.Server.de/Name.txt?", "instanzname");
```

Leider funktioniert dies nur teilweise, da es sein kann, dass der Besucher über einen Proxy-Server auf Ihre Internetseite zugreift. Auf dem Proxy-Server wird die TXT-Datei zwischengespeichert, und man hätte wieder das Problem, dass die TXT-Datei aus einem Cache geladen wird. Man kann den Proxy-Server mit einer HTML-Meta-Angabe anweisen, die Dateien nicht zwischenzuspeichern.

```
<mate-http-equiv="pragma" content="no-cache">
```

Wenn es also um eine TXT-Datei mit dynamischem Inhalt geht, ist es unter Umständen unproblematischer, wenn Sie diese statt dessen über ein CGI einladen.

Eine weiteres Problem beim Einlesen von TXT-Dateien tritt auf, wenn Ihre TXT-Datei so groß ist, dass das Laden einige Zeit in Anspruch nimmt. Hierbei müssten Sie dann unter Flash abfragen, ob die TXT-Datei schon komplett eingeladen ist. Am einfachsten geht das, wenn Sie als letzte Variable in der TXT-Datei ein `&eof=true&` eingeben. Diese Variable können Sie dann unter Flash abfragen.

In Flash 5 ist es möglich, mit onClipEvent(unload) das Abschließen des Ladevorganges abzufragen.

→112 Zugriffzähler(Counter).

Fügen Sie hinter dem Gleichheitszeichen bei Zahlen kein Leerzeichen ein, da Flash dies dann nicht mehr richtig interpretieren kann.

Frame 1

```
eof=false;
loadVariables ("http://www.Server.de/Name.txt", "instanzname");
```

Variable=1 funktioniert also nicht.

Frame 3

```
if (eof eq true) {
    gotoAndPlay (4);
} else {
    gotoAndPlay (2);
}
```

Data.txt

```
&Variable=1&
&Variable2=1&
&eof=true&
```

Tastaturcode
→ *10 Anhang.*

Interessanterweise verhält sich Flash 5 beim Einlesen von TXT-Dateien anders als Flash 4. Wenn in einer TXT-Datei ein String mit einem ↵ vorkommt, enthält der gleiche String anschließend unter Flash 5 einen Zeilenumbruch und eine Leerzeile. Überprüft man das Übermittelte ↵, so stellt man fest, dass Flash 5 dies als die Tastencodes 10 und 13 interpretiert. 10 ist laut der Tastaturcodetabelle *Eingabe*, 13 entspricht einem Wagenrücklauf.

Ordner ExtTextD, Datei: return.fla

Wenn Sie ein *Return* benötigen, benützen Sie den entsprechenden Code aus der URL-Encoding-Tabelle, %0A. Doch sobald man einen längeren Text in eine Textdatei schreibt, wird dieser sehr unübersichtlich. Deshalb haben wir ein kleines ActionScript erstellt, dass das überflüssige Zeichen entfernt. Das Ganze ist als Funktion aufgebaut und universell einsetzbar:

```
function repair (stringalt) {
    string = new String(stringalt);
    for (i=0; i<string.length; i++) {
        if (string.charCodeAt(i) == 13) {
            Teil1 = string.slice(0, i);
            Teil2 = string.slice(i+1, string.length);
            string = Teil1.concat(Teil2);
        }
    }
    return (string);
}
```

Die Funktion hat den Namen *repair* und benötigt einen Parameter. Als Parameter wird der zu überprüfende String übergeben. Der String wird überprüft, so dass die überflüssigen Zeichen nach jedem ↵ entfernt werden und der „reparierte“ String als Funkionswert wieder zurückgegeben wird.

Wenn Sie mit dieser Funktion arbeiten möchten, müssen Sie mit folgendem ActionScript aufrufen:

```
Ihr String = _root.instanzname.repair(IhrString);
```

Durch diesen Befehl wird die Variable `IhrString` an die Funktion `repair` übergeben; dieser String wird dann von der Funktion bearbeitet und als Funktionswert zurückgegeben. Da der Variablen `IhrString` der Funktionswert zugewiesen wurde, wird diese mit dem Wert, den die Funktion zurückliefert, überschrieben. Vorraussetzung ist, dass Sie die Funktion vorher schon eingebunden haben, z.B. durch ein `onclipevent(load)`.

Sieht man sich die Funktion *repair* genauer an, so wird hier erst einmal ein Stringobjekt erzeugt. In Flash 4 standen diese Objekte noch nicht zur Verfügung, aber dennoch konnte man die gleiche Wirkung mit anderen Befehlen erzielen.

Flash 4 Stringbefehle
→ *39 Mailfunktionen.*

Der übergebene Wert (String) bekommt in der Funktion den Namen `Stringalt` zugewiesen. Dies geschieht direkt bei der Deklaration der Funktion `function repair (stringalt){}` . Dadurch wird ein String-Objekt erzeugt mit dem String `Stringalt`. Ein Stringobjekt muss erzeugt werden, um Operationen wie `slice` oder `concat` durchführen zu können. Nachdem das Stringobjekt erstellt und dieses *string* zugewiesen wurde, folgt eine For-Schleife, da man jeden Buchstaben einzeln überprüfen muss. Diese For-Schleife läuft entsprechend der Länge des String, da i bei jedem Schleifendurchlauf um 1 erhöht wird, und die Schleife so lange läuft, bis i größer als die Länge des Strings ist. In der For-Schleife befindet sich eine If-Abfrage; diese überprüft den Code des jeweiligen Zeichens. Die If-Abfrage wird *true*, wenn dieses Zeichen gleich dem Code 13 ist. In diesem Fall werden die Zeichen vor dem i-ten Buchstaben (der gerade überprüft wurde und der den Code 13 besitzt) in die Variable `Teil1` geschrieben und die Zeichen nach dem i-ten Buchstaben in den `Teil2`. Dies geschieht durch den Befehl `slice`:

```
Teil1 = string.slice(0, i);
Teil2 = string.slice(i+1, string.length);
```

Stringname.slice(`anfangsposition`*, endposition)*

Also werden in `Teil1` die Zeichen zwischen o und i geschrieben und in `Teil2` die Werte zwischen i + 1 bis zum Ende. Es heißt i + 1, weil man das ausgewählte Zeichen entfernen möchte. Falls Sie mehrere Zeichen entfernen möchten, müssen Sie zu i eine größere Zahl addieren.

Wir haben hier mit slice und concat gearbeitet, um die Funktion dieser Befehle zu verdeutlichen. An sich wäre es passender gewesen, den Befehl splice zu benutzen.
string.splice(i, 1);

Durch das Trennen des Strings erhält man nun zwei einzelne Strings, die aber das entsprechende Zeichen der i-ten Position des ursprünglichen Strings nicht mehr enthalten.

Mit dem Befehl `concat` kann man die zwei einzelnen Strings wieder zu einem zusammenfügen.

Splice
→ *9 ActionScript-Referenzteil oder 6.1 Memory.*

```
string = Teil1.concat(Teil2);
```

Allgemein ausgedrückt: `NameDesErstenStrings.concat(NameDesZweitenStrings, NameDesDrittenStrings,…..,NameDesX´tenStrings);`

Mit `concat` kann man also beliebig viele Strings zusammenfügen. Normalerweise braucht man das Ergebnis nicht noch einem anderen String zuzuweisen, da alle Strings nach dem Befehl `concat` an den ersten String angehängt werden. Da der String in der Variable `string` in diesem Fall jedoch überschrieben wird, muss man dies eben so lösen. (Die Variable muss überschrieben werden, damit sich die Laufbedingung der for-Schleife automatisch an die neue Länge des Strings anpasst.)

Durch das Trennen des Strings und das anschließende Zusammenfügen kann man das störende Zeichen entfernen. Dieser ganze Vorgang könnte sich wiederholen, falls mehrere Zeichen mit dem Code 13 in dem Text enthalten sind. Die For-Schleife geht alle Zeichen durch und die Zeichen von der If-Abfrage werden daraufhin geprüft, so dass jedesmal, wenn ein Zeichen mit den entsprechenden Spezifikationen gefunden wird, die oben beschriebene Prozedur abläuft. Da das Ganze als Funktion aufgebaut ist und diese einen Wert erzeugt, sollte man diesen Wert auch wieder zurückliefern. Für das Zurückliefern eines Wertes aus einer Funktion ist die Anweisung `return()` zuständig. Durch Eingabe des Arguments `String` gibt die Funktion diesen als Funktionswert zurück.

```
return (string);
```

▶ HTML-Dateien

Ordner ExtTextD, Datei: LadeHTML.fla

HTML-Dateien können nun direkt in Flash abgebildet werden. Man kann sie über folgenden Befehl einladen:

```
loadVariablesNum ("text.html", 0);
```

Die eingelesenen Daten werden in einer Variable auf Level 0 (im `_root`) gespeichert. Diese Variable muss in dem HTML-File vor dem Text stehen, z. B.: Text = (HTML-Text).

Textfelder
→ *43 Textfelder.*

Die Darstellung des HTML-Textes erfolgt in einem entsprechenden Textfeld. Wichtig sind hierbei die Einstellungen: *HTML, Dynamischer Text* und *Mehrere Zeilen*.

Die Farbe sowie die Größe und die Schriftart übernimmt Flash aus dem HTML-File. Auch Links zu anderen Seiten lassen sich so leicht erzeugen. E-Mail-Links mit *Mailto* sind so ebenfalls möglich. Ohne Probleme kann ein Wort unterstrichen, kursiv oder fett geschrieben sein. Hierfür sind folgende HTML-Tags erlaubt:

Tag	Typo-Effekt
 	Linkaufrufe
<B> </B>	fett
<I> </I>	kursiv
<U> </U>	unterstrichen
< font color="#008000"> 	Farbe
 	Schriftart
<font size="+100"> </ font >	Schriftgröße
<p align='left'>	linksbündig
<p align='center'>	mittig
<ü align='right'>	rechtsbündig

Texte als Blocksatz auszurichten ist leider hiermit nicht möglich, genauso wenig kann man direkt Umlaute darstellen oder ein Wort durchstreichen. Die Sonderzeichen aus der URL-Encoding-Tabelle kann man aber dafür verwenden.

Ein HTML-File mit den entsprechenden Modifikationen:

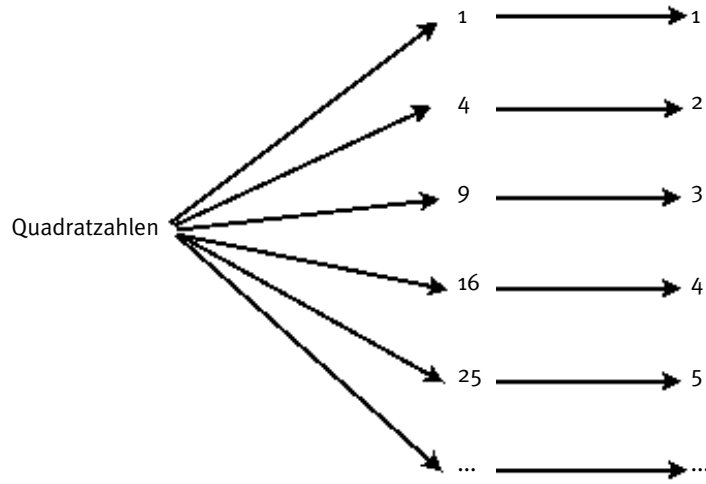
```
HTML=<font color="#FF0000">Dies hier ist der Text aus dem HTML-
File.
Die <font color="#008000">Farbe</font> ,sowie die <font
size="+100">Groesse</font> und die <font face="Braggado-
cio">Schriftart</font> uebernimmt Flash aus dem HTML File.
Auch Links lassen sich so leicht erzeugen :<a href="http://
www.DieFlasher.de"><u>Die Flasher</u></a><font color="#000000">
Ohne Probleme kann auch ein Wort <u>unterstrichen</u>, <i>kur-
siv</i> oder <b>Fett</b> geschrieben sein.
Texte rechtsbuendig oder als blocksatz auszurichten geht leider
nicht, genauso wenig kann man ein Ae, Ue oder Oe darstellen oder
ein Wort durchzustreichen. Die Sonderzeichen aus der URL-Enco-
ding-Tabelle kann man aber auch hier anwenden.
<a href="mailto:e.mai@server.de"><u>E-mail-Links mit Mailto
sind auch moeglich.</u> </a>
</font>
</font>
```

XML

XML.fla

Mit der Erweiterung des ActionScripts um XML ist es in Flash möglich geworden, Daten strukturiert zu übermitteln und in ihnen zu „navigieren“. In dem folgenden Abschnitt stellen wir dar, wie Sie ein XML-Objekt erstellen. Als Beispiel dient eine simple Liste mit Einträgen (Quadratzahlen von 0–24). Haben Sie diese erzeugt, können Sie sie in alle Richtungen nach Einträgen

durchsuchen. Die Hierarchie lässt sich zum besseren Verständnis wie folgt schematisieren:



und wird im ActionScript wie folgt erzeugt:

```
Quadratzahlen = new XML();
Argumente = new XML();
tmpXml=new XML();

for (i=0;i<25;i++){
Ergebnis=Quadratzahlen.createElement('E1 '+i*i);
Ergebnis.attributes.Wert1 = 'Quadratzahl: '+i*i;
Quadratzahlen.appendChild(Ergebnis);
//Hier werden die Quadratzahlen, also die erste Ebene erzeugt.

Argument=Argumente.createElement('E2 '+i);
Argument.attributes.Wert2 = 'Argument: '+i;
Quadratzahlen.lastChild.appendChild(Argument);
//Da im jeweiligen Zyklus der Schleife das neueste 'Kind' von
Quadratzahlen auch gleichzeitig das letzte ist kann man hier die
Daten mit lastChild anhängen
}
tmpXml=_level0.Quadratzahlen.firstChild;
```

Hier wird die Struktur, der Datenstamm, zunächst aufgehängt. Um die Daten nun auszugeben, muss ein Ausgabefeld angelegt werden, in welches die Daten des jeweiligen Knotens eingetragen werden können. In diesem Beispiel heißt das Feld `output2`. Legen Sie auf der Hauptbühne einen Button mit folgendem ActionScript an, um die Daten auszugeben:

```
on(press){
_level0.output2=_level0.Quadratzahlen.firstChild.toString();
}
```

Um in dem „Baum“ ein benachbartes Element zu erreichen, können Sie zwei weitere Schaltflächen mit folgenden Scripten hinzufügen:

```
on(press){
    if(tmpXml.previousSibling!=null){
        _level0.output2=tmpXml.previousSibling.toString();
        tmpXml=tmpXml.previousSibling;
    }
}
```

bzw. für die andere Richtung:

```
on(press){
    if(tmpXml.nextSibling!=null){
        _level0.output2=tmpXml.nextSibling.toString();
        tmpXml=tmpXml.nextSibling;
    }
}
```

Um nun auch noch in den Ebenen zu wechseln, werden nochmals zwei weitere Schaltflächen hinzugefügt, die Gesamtschaltflächenanzahl beträgt nun fünf. Folgende Scripte sind hier nötig:

```
on(press){
    if(tmpXml.parentNode!=null){
        _level0.output2=tmpXml.parentNode.toString();
        tmpXml=tmpXml.parentNode;
    }
}
```

um eine Ebene nach „oben“ zu wechseln, und

```
on(press){
    if(tmpXml.firstChild!=null){
        _level0.output2=tmpXml.firstChild.toString();
        tmpXml=tmpXml.firstChild;
    }
}
```

um eine Hierarchieebene nach unten zu wechseln.

Zu Anfang jeder der Navigationsmöglichkeiten wird überprüft, ob ein nächstgelegenes Element existiert. Damit werden ungültige Werte vermieden.

XML bietet Ihnen die Möglichkeit, Sitemaps damit zu strukturieren, Highscores oder Adressenlisten zu erstellen. Wie diese Daten strukturiert und abgerufen werden, kann im ActionScript-Index nachgeschlagen werden.

Timeout (Screensaver-Effekt)

Wenn Sie einen Countdown oder Ähnliches programmieren wollen, greifen Sie am besten auf die `getTimer();`-Funktion zurück. Diese liefert Ihnen laufend die Zeit in 1ooostel Sekunden, die seit dem Start des Filmes vergangen sind. Wenn Sie einen Screensaver einstellen wollen, greifen Sie auch hier auf dieses Objekt zurück, da es sich hierbei um eine Zeitmessung handelt. Indem Sie laufend die Differenz zwischen der letzten Unterbrechung (z.B. Mausbewegung) und der insgesamt verstrichenen Zeit berechnen und diese mit Ihrem Zeitlimit vergleichen, können Sie den gewünschten „Screensaver-Effekt" erzeugen. In unserem Beispiel erstellen wir eine Filmsequenz auf der Hauptbühne sowie ein dynamisches Textfeld mit der Bezeichnung „output". Nun füge ich folgendes ActionScript in die Filmsequenz ein:

```
onClipEvent(load){
a=0;
}
onClipEvent(mouseMove){
a=getTimer()/1000;
}
onClipEvent(enterFrame){
    sec = int(getTimer()/1000-a);
    if(Number(sec) > 10){
        _level0.output='Zeit vorbei!';
    } else {
        _level0.output=sec;
    }
}
```

Wie zu sehen ist, wird zu Anfang ein Wert *a* initialisiert, der laufend die Differenz zwischen der letzten Mausbewegung und der insgesamt verstrichenen Zeit anzeigt. Wird die Maus bewegt, wird der Wert von *a* auf die aktuell verstrichene Zeit erhöht und die Zeitdifferenz ist wieder *o*. Überschreitet die Zeitdifferenz 10 Sekunden, so wird der Text *Zeit vorbei!* ausgegeben.

Eine solche Zeitmessung ist natürlich auch bei der Programmierung von Spielen von Interesse, wie sie in Kapitel 6 beschrieben wird.

Erstellen von Smartclips

Ordner erstellSC, Datei: maquee.fla

Im folgenden wird die Erstellung von Smartclips beschrieben, das Ändern von Variablen und der Entwurf einer benutzerdefinierten Smartclip-Oberfläche für die Definition der Smartclipvariablen.

Berücksichtigen Sie bei der Erstellung eines Smartclips die Variablen, die der Benutzer später ändern kann, da das komplette ActionScript entsprechend ausgelegt sein muss.

Smartclips und wozu man diese benutzt, wird in → *92 Erstellen von Smartclips* *beschrieben.*

▶ Aufbau eines Smartclips

Für einen Lauftext (maquee) erstellen Sie zuerst mit Hilfe eines Textfeldes und eines ActionScript einen Movieclip für den Smartclip. (Wie man den Movieclip in einen Smartclip umwandelt, wird später beschrieben.) In diesem Movieclip erstellt man am besten noch einen anderen Movieclip, um dort die `onclipEvents` einfügen zu können. In diesem Movieclip positioniert man dann ein dynamisches Textfeld mit der Variable `textausgabe`. Die Schriftart sollte einen gleichbleibenden Zeichenabstand haben wie z.B. `_tpewriter`. Meistens werden Smartclips in einem zusätzlichen Movie erstellt, um eine Möglichkeit zu haben, die `onClipEvent`-Befehle einzufügen.

▶ Erstellen eines Smartclips

Nachdem man die zwei Movies erstellt hat, kehrt man auf die Hauptbühne zurück, öffnet die Bibliothek Strg + L , klickt mit der rechten Maustaste auf das Movie, welches man zu Beginn für das Smartclip erstellt hat, und auf SEQUENZPARAMETER DEFINIEREN.

Mit dem Plus-Button können Sie Parameter hinzufügen, mit dem Minus-Button wieder entfernen. Mit den Pfeil-Buttons rechts oben in der Ecke können Sie den ausgewählten Parameter nach unten oder oben verschieben. Dies hat auf die Übergabe der Werte keinen Einfluss, sondern dient nur dazu, dies für den späteren Benutzer logisch anzuordnen. Unter „Name“ muss man den Namen der `Variablen` eingeben, bei „*Wert*“ ihren Wert und bei „Typ“ kann man zwischen *Default, Array, Objekt* und List wählen.

- **Default:**

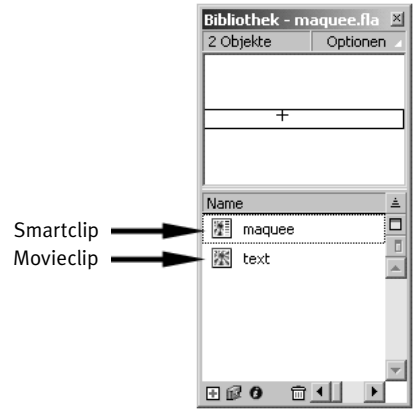
Eine normale Variable wird dadurch an den Smartclip übergeben. Diese kann einen Wert oder einen String enthalten.

- **Array:** Dient der Übergabe von Werten, die in einem Array liegen. Der User kann alle Werte beliebig verändern.

- **Objekt:** Übergibt ein Objekt an den Smartclip. Dieses enthält Variablen.
- **List:** Ist eine Variable, die nur bestimmte Werte annehmen kann: Man kann dadurch z. B. dafür sorgen, dass der Benutzer der Variablen `wochentage` nur die Werte `:Mo, Di, Mi, Do, Fr, Sa` oder `So` zuordnen kann.

Standard-Offset ist die Einstellung, die zu Beginn angezeigt wird.

Nachdem man bei den Sequenzparametern Variablen definiert hat, wird der Movieclip als Smartclip in der Bibliothek angezeigt.



Bei den *Sequenzparametern* unter VERKNÜPFUNG MIT BENUTZERDEFINIERTER OBERFLÄCHE kann man den Pfad zu einem SWF-File eingeben, dass dann später unter den *Sequenzparametern*, statt der normalen Anzeige dargestellt wird.

Nachdem man die Sequenzparameter eingestellt hat, kann man das ActionScript erstellen:

```
onClipEvent (load) {
    text = _parent.text;
    geschwindigkeit = _parent.geschwindigkeit;
    geschwindigkeitsindex = geschwindigkeit;
    textstring = new String(text);
    textstring = textstring+"   ";
    textlaenge = textstring.length;
    for (i=textlaenge; i<40; i += textlaenge) {
    textstring = textstring+textstring;
    }
    i = 0;
    textlaenge = textstring.length;
}
onClipEvent (enterFrame) {
    geschwindigkeitsindex--;
    if (geschwindigkeitsindex<1) {
        geschwindigkeitsindex = geschwindigkeit;
        i++;
        if (i == textlaenge) {
        i = 0;
        }
        textausgabe = textstring.substring(i,
            textlaenge)+textstring.substring(0, i);
    }
}
```

Mit den zwei Befehlen `text = _parent.text;` und `geschwindigkeit = _parent.geschwindigkeit;` werden die entsprechenden Variablen aus dem darunter liegenden Movie (Smartclip) in das aktuelle Movie kopiert. Anschließend erfolgt die Bearbeitung des Textes. Zuerst wird ein Stringobjekt erstellt `textstring = new String(text);` danach werden diesem String zwei Leerzeichen angehängt `textstring = textstring+"   ";`. Danach wird die Länge des Strings abgefragt; falls dieser kleiner als 40 Zeichen ist, wird der gleiche String noch einmal an den String angehängt, damit die Gesamtlänge des Textfeldes auch immer mit Text gefüllt wird.

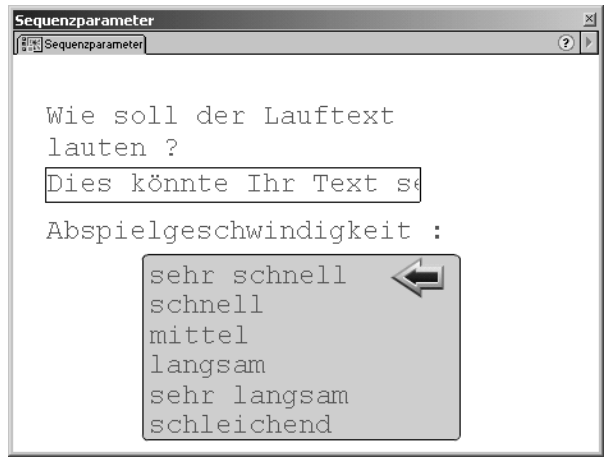
Bei den Befehlen im `onClipEvent (enterFrame)` wird der String zerlegt, abhängig vom Wert von `i`, so dass immer zuerst die Buchstaben ab i bis zum Ende `textstring.substring(i, textlaenge)` und danach die Buchsta-

ben vom Beginn bis `i` `textstring.substring(0, i)` die Textausgabe bilden. Somit wäre der Smartclip fertig erstellt.

▶ Benutzerdefinierte Oberfläche

Ordner erstellSC, Datei: bdo.fla

Unter den Sequenzparametern bei *„Verknüpfung mit benutzerdefinierter Oberfläche“* kann man den Pfad zu einem SWF-File eingeben, das dann später unter den *Sequenzparametern* statt der normalen Anzeige dargestellt wird. Diese Eingabemaske könnte beispielsweise für den Lauftext folgendermaßen aussehen:



Das erstellte SWF kann wie ein normales SWF aufgebaut sein. Es muss allerdings alle Variablen aus dem Movie mit dem Instanznamen XCH beziehen. Deshalb ist es ratsam, alles direkt in dieses Movie zu verlagern. Die somit übergebenen Variablen sind in dem Smartclip direkt verfügbar. Bei einem Button, der die Geschwindigkeit auf einen bestimmten Wert setzt, würde beispielsweise das ActionScript so aussehen:

```
on (release) {
    geschwindigkeit = 2;
}
```

Dort wird also nur die entsprechende Variable auf einen Wert gesetzt, Pfadangaben zu anderen Movies oder Ähnliches sind nicht notwendig.

Achten Sie darauf, dass sich das SWF für die Sequenzparameter im selben Verzeichnis wie das Movie mit dem Smartclip befindet. Ansonsten müssen Pfade angegeben werden, die eventuell bei anderen Rechnern nicht mehr funktionieren. Auch wenn Flash bei *„Verknüpfung mit benutzerdefinierter Oberfläche“* einen absoluten Pfad angibt, sind auch relative Pfadangaben möglich.

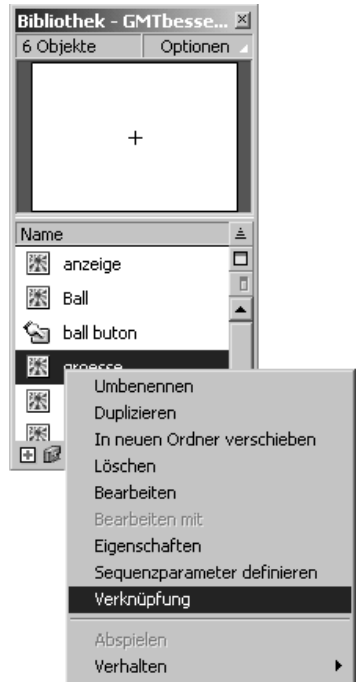
Shared Library

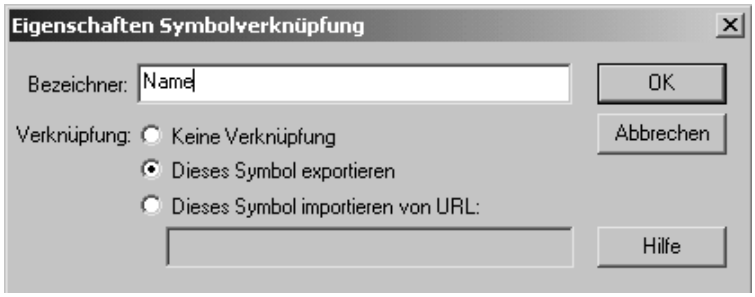
In Flash 5 ist es möglich, Objekte aus anderen SWFs zu importieren. Wie das Exportieren und Importieren von Objekten funktioniert, wird hier erläutert.

In Flash 5 gibt es nun die Möglichkeit, einzelne Elemente eines Films zu exportieren und so für andere Filme verfügbar zu machen. Dies bietet den Vorteil, immer wieder auf die gleichen Elemente zugreifen zu können, die der User schon eingeladen hat, und das Objekt nur einmal aktualisiert werden muss.

Um ein Objekt für andere Benutzer freizugeben, klicken Sie es in der Bibliothek mit der rechten Maustaste an und wählen den Menüpunkt „*Verknüpfung*“ aus.

Wählen Sie in der Dialogbox „Dieses Symbol exportieren“ und geben bei „*Bezeichner*“ einen Namen ein.





Über diesen Weg können Sie in einem Flashmovie die Bibliothek (Library) eines anderen Movies über DATEI › ALS GEMEINSAME BIBLIOTHEK ÖFFNEN öffnen und per Drag & Drop das entsprechende Objekt in die Bibliothek importieren.

Drag &Drop von Objekten aus der Bibliothek

→ *74 Soundeffekte.*

In der Bibliothek, in die das Objekt importiert wurde, werden die Verknüpfungsparameter automatisch eingestellt.

Dies setzt voraus, dass beide SWFs im gleichen Verzeichnis liegen.Wenn Sie die Files später auf einen Server legen, ist es günstiger, direkt mit absoluten URLs zu arbeiten.

Wenn Sie auf der Festplatte testen, benutzen Sie relative Pfadangaben, wobei Sie aber darauf achten sollten, in den Verzeichnisnamen keine Sonderzeichen oder Überlängen zu verwenden.

5.4 Flash und JavaScript

Flash-Formular (Flash 4)

Um ein Formular in Flash zu realisieren, muss man auf JavaScript zurückgreifen. Hierzu muss Ihr Internetprovider serverseitige Scripts zulassen. Wir besprechen hier ein Beispiel für die Frontpage-Extension, das in der Praxis recht verbreitet ist. Das Script für ein Formular, das ein anderes (z.B. CGI) Script anspricht, müssen Sie ggf. bei Ihrem jeweiligen Provider erfragen.

Zuerst wird eine Eingabemaske in Flash erstellt; alle Textfelder, die der Eingabe dienen, müssen dynamisch sein und einen Variablennamen haben. Der Button „Löschen“, den jedes Formular haben sollte, könnte wie folgt aussehen:

```
on (press) {
    name = "";
    adresse = "";
    telefon = "";
    email = "";
    firma = "";
    frage = "";
}
```

Je nachdem wie Sie die Eingabefelder benannt haben und wie viele Positionen Ihr Formular enthält, können Sie es variieren. Um die Daten an das JavaScript zu übermitteln, muss man folgendermaßen vorgehen: Die Datenübertragung von Flash an das Script und die Datenübermittlung vom Client

an den Server muss getrennt erfolgen, da sonst ein leeres Datenblatt versandt wird:

```
on (press) {
    pos1 = name;
    pos2 = adresse;
    pos3 = telefon;
    pos4 = email;
    pos5 = firma;
    pos6 = frage;
    if (name="" or telefon.length  lt 5 …) {
        gotoAndPlay („unvollstaendig");
    }
    getURL ("javascript:flashtojs(" add name add ", " add
adresse add … add Telefon add ")");
}
on (release) {
    getURL ("javascript:uebermittle()");
    }
}
```

Die entsprechende HTML-Datei mit dem zusätzlichen JavaScript sieht wie folgt aus:

```
<script language="javascript">
Function flashtojs(name,adresse,telefon, ...){
Document.form[0].Name=name;
Document.form[0].Adresse=adresse;
Document.form[0].Telefon=telefon;
Document.form[0].eMail=eMail;
:
}
Function uebermittle(){
Document.form[0].submit();
}
</script>
```

Das Script ist individuell anzupassen, genauso wie man den WEBBOT-Tag am besten nochmals von Frontpage erzeugen lässt:

```
<form method="POST" action="--WEBBOT-SELF--">
```

Hier gibt Frontpage mit `method=` die Transferart an:

```
  <!--webbot bot="SaveResults" U-File="fpweb:///_private/
form_results.txt"
```

Der Pfad zur Erstellung des Formular-Logs:

```
S-Format="TEXT/CSV" S-Label-Fields="TRUE" B-Reverse-Chrono-
logy="FALSE" S-Email-Format="TEXT/PRE" S-Email-Address=mein-
name@meine-domain.de
```

Ersetzen Sie an dieser Stelle mein-name@meine-domain.de durch Ihre individuelle E-Mail-Adresse.

```
B-Email-Label-Fields="TRUE" S-Builtin-Fields
S-Form-Fields="Name Telefon Adresse email Firma " -->
```

Mit `S-Form-Fields=` werden für das Feedback relevante Angaben deklariert.

```
<input type="hidden" name="Name">
<input type="hidden" name="Telefon">
<input type="hidden" name="Adresse">
<input type="hidden" name="email">
<input type="hidden" name="Firma">
```

Die `input`-Tags beinhalten das für den Benutzer unsichtbare Formular, an das Flash die Werte aus dem Film mittels eines JavaScript-Aufrufs übermittelt.

```
</form>
```

Sie füllen im Endeffekt per Flash ein unsichtbares HTML-Formular aus und verschicken es auch von dieser HTML-Seite aus.
Natürlich wäre es noch schöner, ein „mailto:mein-name@meine-domain.de“ in der

```
<form action="..">
```

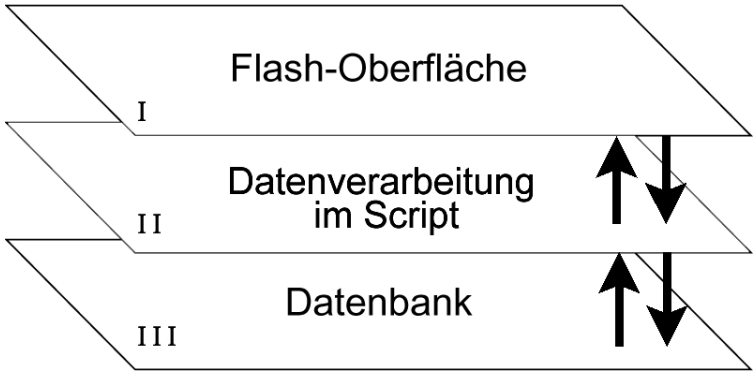
zu haben, doch leider ignoriert der Browser Netscape den JavaScript-Befehl

```
Document.form[0].submit();
```

völlig. Wer keinen Wert auf Netscape-Besucher legt, sei an dieser Stelle auf diese Möglichkeit hingewiesen.

Datenkommunikation zwischen Flash und JavaScript

Ein Datenaustausch zwischen Flash und JavaScript ist für viele Zwecke sinnvoll, da Flash 5 zwar XML unterstützt, jedoch die meisten Internetseiten noch per Java- oder VisualBasic-Script auf SQL-Datenbanken zugreifen. Flash bildet bei den meisten Seiten die oberste Zugriffsebene. Datenbankzugriffe, Währungsumrechnungen etc. werden oft von der Seite, die dahinter in einer unteren Ebene liegt, bewältigt. Dies ist dann sinnvoll, wenn man z.B. für einen Warenkorb in Flash feste Schnittstellen geschaffen hat (Anzahl, Art, Format der abzubildenden Werte) und dadurch das Script schneller modifiziert werden kann.



Die Ebene I ist in diesem Fall der Flash-Film, der in direktem Kontakt zum Anwender steht.

Die Ebene II beinhaltet z.B. das Frameset mit den notwendigen JavaScripts zur Filterung der Anfragen und Ergebnisse. Von dieser Ebene aus werden auch die clientseitigen Anfragen an den Server gestellt.

Ebene III ist meist komplett serverseitig, wenn es sich wie z.B. bei Shops um große Datenbestände handelt. Bei geringeren Datenmengen, z.B. zur Navigation einer umfangreichen Sitestruktur, kann es durchaus sinnvoll sein, die Datenbank direkt auf den Client zu übertragen, da Datenbankanfragen den Server belasten und Traffic und damit zusätzliche Wartezeit für den Benutzer erzeugen.

Folgend möchten wir auf eine Möglichkeit eingehen, zwischen Flash und JavaScript Daten auszutauschen. Wir gehen davon aus, dass die Variable `/:input` der Flashdatei Warenkorb.swf aus einem externen Script geändert werden soll.

```
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://active.macromedia.com/flash2/cabs/
swflash.cab#version=4,0,0,0"
 ID= Warenkorb WIDTH=750 HEIGHT=500>
```

Die Angabe der `ID=Warenkorb` ist zu beachten!

```
 <PARAM NAME=movie VALUE=" Warenkorb.swf">
 <PARAM NAME=quality VALUE=best>
 <PARAM NAME=scale VALUE=exactfit>
 <PARAM NAME=salign VALUE=LT>
 <PARAM NAME=bgcolor VALUE=#FFFFFF>
 <EMBED src=" Warenkorb.swf" quality=best scale=exactfit
salign=LT bgcolor=#FFFFFF  WIDTH=750 HEIGHT=500 TYPE="applica-
tion/x-shockwave-flash" swLiveConnect=true
```

`SwLiveConnect=true` muss angegeben werden, sonst funktioniert die Kommunikation in Netscape nicht.

```
PLUGINSPAGE="http://www.macromedia.com/shockwave/download/
index.cgi?P1_Prod_Version=ShockwaveFlash"></EMBED>
</OBJECT>
```

Damit ist der `<object>` -Tag modifiziert, nun muss noch ein entsprechendes JavaScript eingefügt werden, um Werte in Flash zu ändern.

```
<script language="javascript">
var InternetExplorer = navigator.appName.indexOf("Microsoft")
!= -1;
var path = InternetExplorer ? Warenkorb : document.embeds[0];
```

Hier wird die ID `Warenkorb` angesprochen, ggf. muss der Wert o von `document.embeds[0]` in 1, 2 ,3, ... geändert werden, wenn Sie innerhalb dieses Dokumentes mehrere Flash-Filme eingebunden haben.

```
path.SetVariable('input', this.window.name);
```

Hier wird der Wert übergeben, in diesem Fall die Bezeichnung des Fensters, in dem sich das JavaScript befindet. Es ist aber auch möglich, `this.window.name` durch `'Hello World'` zu ersetzen.

```
</script>
```

Diese Variante per `GetUrl` aufzurufen, eignet sich leider nur bedingt, da sie sehr viel Rechnerleistung benötigt und in einer sich laufend wiederholenden Abfrage (z.B. zur automatischen Aktualisierung) das System fast zum Erliegen bringt. Es empfiehlt sich die zweite Variante. Diese veranschaulichen wir anhand einer Ladestatus-Abfrage (→ *Preloader*).

```
<script language="JavaScript">
<!
```

Folgend wird die Funktion zur Ladestatusabfrage deklariert:

```
function status()
{
 InternetExplorer = navigator.appName.indexOf("Microsoft") != -
1;
 path = InternetExplorer ? Warenkorb : document.embeds[0];
```

Mittels `PercentLoaded()` wird die bisher übertragene Datenmenge prozentual gemessen am Object `path`, also unserem Flash-Film.

```
 perc = path.PercentLoaded();
```

Hier wird mittels `SetVariable` der Wert an Flash übermittelt:

```
 path.SetVariable('input',perc);
}
```

Damit die Abfrage laufend wiederholt wird, empfiehlt sich nur der Aufruf der Funktion mit `FS_Command`. `GetUrl` scheidet aufgrund einer zu hohen CPU-Belastung aus.

```
var InternetExplorer = navigator.appName.indexOf("Microsoft")
!= -1;
function Warenkorb_DoFSCommand(command, args) {
var WarenkorbObj = InternetExplorer ? Warenkorb : docu-
ment.Warenkorb;
```

Abfrage des Kommandos, das in Flash aufgerufen wurde. Die If-Bedingung führt, wenn das Kommando `pruefe` lautet, die Funktion `status()` aus, die wiederum aktuelle Daten an Flash übermittelt.

```
  if (command == "pruefe")
      {status();}
}
```

Der von Flash bei der Veröffentlichung mit der Einstellung „Flash mit FS Command“ automatisch generierte Tag:

```
if (navigator.appName && navigator.appName.indexOf("Micro-
soft") != -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 Warenkorb_FSCommand(ByVal command, ByVal
args)\n');
    document.write('  call Warenkorb_DoFSCommand(command,
args)\n');
    document.write('end sub\n');
    document.write('</SCRIPT\> \n');
}
//-->
</SCRIPT>
```

An dieser Stelle können Sie diesen Wert, den Sie nach Ausführung des JavaScripts in Flash zur Verfügung haben, beliebig weiterverarbeiten.

Jetzt wissen Sie, wie man Werte von JavaScript an Flash schickt. Folgend möchte ich Ihnen eine Möglichkeiten veranschaulichen, Daten aus Flash an JavaScript zu übermitteln. Die genannte Möglichkeit mittels `GetUrl`

```
getURL ("javascript:flashtojs(" add name add ", " add adresse
add … add Telefon add ")");
```

ist nur im Einzelfall anzuwenden; es ist besser, weiterhin mit `FS_Command` zu arbeiten. Wie Sie dem Script zur Datenvermittlung von JavaScript an Flash bereits entnehmen können, können auch Argumente übergeben werden. Die entsprechende Variable lautet im obigen JavaScript `args` und kann direkt so im JavaScript weiterverarbeitet werden.

Der Erdbebeneffekt

Zu einem netten Effekt, den man durch geschicktes Timing im Zusammenspiel von Flash und JavaScript erzeugen kann, gehört auch der Erdbebeneffekt. Im folgenden Abschnitt gehen wir zuerst auf das dazugehörige JavaScript ein, danach werden Einsatzmöglichkeiten erläutert.

```
<script language="javascript">
```

Hier wird die Funktion deklariert; `Beben` der Name, `amp` und `len` sind jeweils die Parameter.

`Get URL ("javascript:Beben(10,10)")` ist der entsprechende Aufruf aus dem ActionScript. Der erste Parameter bestimmt die Stärke des Bebens, der zweite die Dauer:

```
function Beben(amp, len) {
```

Um Fehler zu vermeiden, wird überprüft, ob die Funktion vom Browser unterstützt wird:

```
    if (self.moveBy) {
```

Nun wird eine Schleife eingeleitet, die vom Parameterwert `amp` bis 0 herunterzählt.

```
        for (x = amp; x >= 0; x--) {
```

An dieser Stelle wird eine zweite Schleife gestartet, die die erste verzögert, so dass der Inhalt dieser Schleife für jeden Wert von `amp` `len`-mal durchlaufen wird.

```
            for (i = len; i >= 0; i--) {
                self.moveBy(0,x);
                self.moveBy(x,0);
                self.moveBy(0,-x);
                self.moveBy(-x,0);
            }
        }
    }
}
</script>
```

Fullscreen

Es gibt die Möglichkeit, eine Website im Vollbildschirmmodus oder kurz „Fullscreen“ darzustellen. Allerdings muss dies in Flash wie auch in JavaScript und HTML entsprechend angelegt sein. Zu Beginn des Filmes muss der Befehl

```
FS Command ("fullscreen", "true")
```

eingesetzt werden; das hat für den Stand-alone-Player sofortige Gültigkeit, und solange Sie den Vollbildschirmmodus nur für eine interaktive CD benö-

tigen reicht das auch. Wollen Sie jedoch den Flash-Film im Internet veröffentlichen, ist noch Folgendes notwendig. Die Seite, auf die Ihr bereits mit dem `FS_Command` präparierter Flash-Film steht, muss nun mittels eines JavaScripts aufgerufen werden. Der Aufruf sieht wie folgt aus:

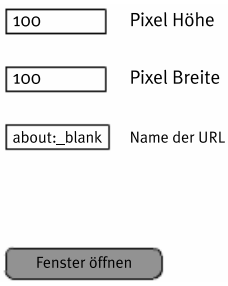
```
<script language="JavaScript">
window.open("Zielfilm.htm","Fullscreen","fullscreen=yes");
</script>
```

Jetzt wird der komplette Bildschirm von Ihrem Flash-Film ausgefüllt.

Dabei müssen Sie allerdings darauf achten, dass die Seitenverhältnisse gewahrt bleiben. Wenn Ihr Flash-Film ein anderes Format aufweist als 4:3, werden Inhalte teilweise verzerrt angezeigt. Ab einer Auflösung von 1024 x 1280 Pixel ändert sich das Seitenverhältnis.

Fenster in bestimmter Größe öffnen

Wenn Sie aus Flash heraus ein Popup erzeugen möchten, um z. B. eine Partnerseite anzuzeigen, müssen Sie auf ein JavaScript in der HTML-Datei zurückgreifen. Das folgende Beispiel zeigt, wie Sie ein Fenster mit bestimmten Maßen aus Flash öffnen können.



Windowopen.fla

Das erste Textfeld bekommt den Variablennamen `height`, das zweite `width` und das dritte `toUrl`. Die Möglichkeit, die Felder zu bezeichnen, haben Sie unter TEXTOPTIONEN Strg+T, Register TEXTOPTIONEN. Allerdings muss auch die Eigenschaft „Texteingabe“ aus dem obersten Dropdown-Feld des gleichen Registers ausgewählt sein.

Nun ordnen Sie dem Button „FENSTER ÖFFNEN“ folgendes ActionScript zu:

```
on (press) {
    getURL
("javascript:oeffne('"+width+"','"+height+"','"+tourl+"');",
"_self");
}
```

und passend zu dem Aufruf in die HTML-Seite des Filmes das JavaScript:

```
<script language="javascript">
function oeffne(iWidth,iHeight,sUrl){
window.open(sUrl,"","width="+iWidth+",height="+iHeight);
}
</script>
```

Natürlich kann diese Funktion für Ihre Zwecke modifiziert werden, zumal Sie sicher keinem User zumuten wollen, die URLs durch geschicktes Raten auszuwählen.

5.5 CGI – das Common Gateway Interface

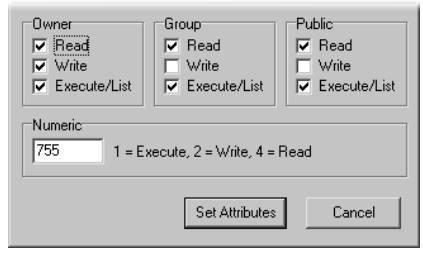
Wenn Sie im Internet surfen, stoßen Sie früher oder später auf ein CGI. Auf den ersten Blick werden Sie dies vielleicht gar nicht bemerken, da CGIs immer in Verbindung mit anderen Elementen auftreten. Aber sobald eine Datenbank abgefragt/bearbeitet wird, ein Zähler (Counter) die Anzahl der Besucher zählt, Sie sich in ein Gästebuch eintragen oder in einem Forum einen Beitrag schreiben, kann dies mit einem CGIs erstellt sein. Und Flash kann mit CGI-Scripten kommunizieren. Wir können auf CGIs hier nur kurz eingehen. Sie werden erfahren, was Sie ändern müssen, um zum gewünschten Ergebnis zu kommen. Darauf, welcher Befehl in einem CGI welche Funktion hat, kann an dieser Stelle hingegen nicht genauer eingegangen werden.

Wir beschreiben nur einige wenige CGIs. Da CGI-Scripte sehr vielseitig sind, kann man damit aber schon viele der benötigten Funktionen erstellen.

Die beschriebenen CGIs erweisen sich auf den ersten Blick als unhandlich, haben aber den Vorteil, flexibel einsetzbar zu sein, so dass die meisten Eigenschaften eines CGIs auch von Flash aus geändert werden können, ohne direkt den Quelltext des CGIs umschreiben zu müssen.

Wenn Sie an dem CGI etwas ändern, müssen Sie darauf achten, dass Sie dieses anschließend wieder im Unix-Format abspeichern.

Beim Hochladen auf den Server müssen Sie dem CGI die Attribute (chmod) 755 zuweisen. Wie dies zu bewerkstelligen ist, variiert zwischen den verschiedenen FTP-Programmen. Bei den meisten Programmen geht dies jedoch durch einen Klick mit der (rechten) Maustaste auf das entsprechende auf dem Server liegende CGI und der anschließenden Auswahl von SET ATTRIBUTES.



Es kommt auch vor, dass Sie ein TXT-File vorher erstellen und diesem die Attribute 666 zuordnen müssen, da die CGIs auf dem Server keine TXT-Files erzeugen können.
Manche Server akzeptieren die Endung *.cgi nicht, in diesem Fall müssen Sie das CGI in *.pl umbenennen.

Sie müssen beim Uploaden der TXT-Datei darauf achten, diese im ASCII-Modus hochzuladen.

Falls Ihr CGI nicht auf Anhieb läuft, ist es meistens nützlich, mit einem getUrl ein neues Fenster zu öffnen und darin das CGI zu starten, anstatt die Variablen mit `loadVariables` an das CGI zu übergeben. Wenn man dabei die Variablen mit GET verschickt, werden diese dann in der URL mit angezeigt, so dass man eine direkte Kontrolle hat. Allerdings funktioniert dann die Rückgabe der Werte aus dem CGI an Flash nicht mehr, da das CGI diese in das neue Fenster schreibt.

Versenden einer E-Mail

Ordner cgiEmail, Dateien: sendmail.fla bzw. sendmail.cgi

Wenn Sie einen Server haben, der CGI unterstützt, und eine E-Mail versenden möchten, ohne den Standard-E-Mail-Client des Besuchers zu nutzen, rufen Sie ein CGI von Flash aus auf und verschicken die E-Mail automatisch.

```
loadVariables ("http://Server.de/cgi-bin/sendmail.cgi", "", "POST");
```

Mit diesem Befehl werden alle Variablen, die sich im `_root` bzw. `_level0` befinden, an das CGI geschickt. Dadurch wird auch das `sendmail.cgi` gestartet.

Die mit zu übergebendenVariablen heißen `email` (diese enthält den Namen des Absenders), `Subjekt` (Betreff der E-Mail) und `Body` (Inhalt der E-Mail).

Man könnte auch anstatt einer einfachen E-Mail unter Flash ein Eingabeformular erstellen und mittels dieses CGIs die Daten verschicken.

Das CGI sieht wie folgt aus:

```
#!/usr/bin/perl

###      Konfigurationsvariablen        ###
#############################################
$datvrz = "/www/server.de/cgi-bin";     ### absoluter Pfad zur
ssi-count.dat
$mailprog = '/usr/sbin/sendmail';
$sendean = "E-Mailadresse\@Server.de";
###            Config-Ende              ###
#############################################

$Data = &CGIreadData;
%Data = &CGIprepData ($Data);
&send_mail;

sub send_mail
{
open (MAIL, "|$mailprog -t") || print "E-Mail-Programm konnte
nicht gestartet werden";
                  print MAIL "To: $sendean\n";
                  print MAIL  "From: $Data{'email'}\n";
                  print MAIL  "Subject: $Data{'subjekt'}\n\n";
                  print MAIL "$Data{'body'}\n";
close (MAIL);
}

sub CGIreadData
{
  local $cgiData;
  if ( $ENV{REQUEST_METHOD} eq "POST" )
```

```
  { read ( STDIN , $cgiData , $ENV{'CONTENT_LENGTH'} ); }
  else
  { $cgiData = $ENV{QUERY_STRING}; }
  return $cgiData;
}

sub CGIprepData
{ local ($cgiDataskalar, $name, $Data);
  local @cgiDatalist;
  local %cgiDatahash;
  if ( $_[0] ) {
    $cgiDataskalar = $_[0];
  }
  else {
    print STDERR ("Es wurden keine Data empfangen\n");
  }
  @cgiDatalist = split(/[&;]/ , $cgiDataskalar);
  foreach $valuelist (@cgiDatalist)
  {
    $valuelist =~ s/\+/ /go;
    ($name, $Data) = split( /=/ , $valuelist );
    $name =~ s/\%(..)/pack("c",hex($1))/ge;
    $Data =~ s/\%(..)/pack("c",hex($1))/ge;
    $cgiDatahash{$name} = $Data;
  }
  return %cgiDatahash;

}
```

Im Kopf werden die Variablen gesetzt, die Sie noch anpassen müssen.

```
$datvrz = "/www/server.de/cgi-bin";
```

Hier muss die URL zu Ihrem cgi-bin-Verzeichnis stehen. Falls Ihr Server ein solches Verzeichnis nicht besitzt, können Sie davon ausgehen, dass dieser Server CGIs nicht unterstützt.

```
$mailprog = '/usr/sbin/sendmail';
```

Hier steht der relative Pfad zu dem Sendmail-Verzeichnis. Im Normalfall ist dieser Pfad richtig. Falls das CGI so nicht funktioniert, erkundigen Sie sich bei Ihrem Webspace-Anbieter nach dem Pfad zu Ihrem Sendmail-Verzeichnis.

```
$sendean = "E-Mailadresse\@Server.de";
```

Dies ist die E-Mail-Adresse, an die die E-Mail geschickt werden soll. Da dieses Script für ein Eingabeformular gedacht ist, wird die E-Mail immer an dieselbe Adresse geschickt, die hier angegeben werden muss.

Die restlichen Punkte des CGI-Scripts sind nicht mehr so wichtig. Ich will aber trotzdem kurz ihren Zweck erläutern:

```
$Data = &CGIreadData;
```

Dies setzt die Variable `Data` gleich dem Ergebnis des Funktionsaufrufes `CGIreadData`.

```
%Data = &CGIprepData ($Data);
```

Hier wird `Data` zuerst an die Funktion `CGIprepData` übergeben und anschließend wird das Ergebnis dieser Funktion wieder `Data` zugeordnet.

```
&send_mail;
```

Hier wird die Unterfunktion `send_mail` aufgerufen.

```
sub CGIreadData
{
  local $cgiData;
  if ( $ENV{REQUEST_METHOD} eq "POST" )
  { read ( STDIN , $cgiData , $ENV{'CONTENT_LENGTH'} ); }
  else
  { $cgiData = $ENV{QUERY_STRING}; }
  return $cgiData;
}
```

Dies ist die erste aufgerufene Unterfunktion `CGIreadData`. Sie überprüft, ob die Daten mit POST oder GET gesendet wurden und speichert diese entsprechend als String in der Variablen `$cgiData`. Diese Variable wird dann als Funktionswert zurückgegeben.

Danach wird die zweite Unterfunktion `CGIprepData` gestartet. Diese überprüft zu Beginn, ob die Eingabe überhaupt Daten enthält.

```
sub CGIprepData
{ local ($cgiDataskalar, $name, $Data);
  local @cgiDatalist;
  local %cgiDatahash;
  if ( $_[0] ) {
    $cgiDataskalar = $_[0];
  }
  else {
    print STDERR ("Es wurden keine Data empfangen\n");
  }
```

Nach dieser Überprüfung wird der übermittelte String in seine einzelnen Bestandteile zerlegt. Anschließend wird der zurückgegebene Funktionswert, `cgiDatahash`, `Data` zugeordnet. So kann man darüber im restlichen CGI die von Flash übermittelten Variablen aufrufen.

```
  @cgiDatalist = split(/[&;]/ , $cgiDataskalar);
  foreach $valuelist (@cgiDatalist)
  {
    $valuelist =~ s/\+/ /go;
    ($name, $Data) = split( /=/ , $valuelist );
    $name =~ s/\%(..)/pack("c",hex($1))/ge;
    $Data =~ s/\%(..)/pack("c",hex($1))/ge;
    $cgiDatahash{$name} = $Data;
  }
  return %cgiDatahash;
}
```

Nachdem nun die übermittelten Variablen überprüft und verfügbar gemacht wurden, kann man die E-Mail versenden. Dies geschieht durch der Funktion `send_mail`:

```
sub send_mail
{
open (MAIL, "|$mailprog -t") || print "E-Mail-Programm kann
nicht gestartet werden";
                    print MAIL "To: $sendean\n";
                    print MAIL  "From: $Data{'email'}\n";
                    print MAIL  "Subject: $Data{'subjekt'}\n\n";
                    print MAIL "$Data{'body'}\n";
close (MAIL);
}
```

Falls Sie weitere Variablen übermitteln, können Sie diese mit der Definition `$Data{'Variable'}`. im CGI-Script ansprechen

Schreiben und Lesen von Textdateien

Ordner Counter, Datei: counter.cgi sowie Ordner Gastbuch, Datei: gb.cgi

Es gibt sicherlich bessere CGIs, die direkt für den jeweiligen Zweck geschrieben wurden. Da dieses CGI vielseitig einsetzbar sein soll, ist es natürlich nicht immer das komfortabelste. Dafür kann man mit diesem CGI in eine beliebige Textdatei schreiben bzw. eine beliebige Datei erzeugen oder aber Daten aus einer beliebigen Textdatei auslesen.

Das CGI braucht die Variable `name`, welche den Namen der TXT-Datei enthalten muss. Zudem muss diesem CGI mit der Variablen `aktion` mitgeteilt werden, was es machen soll. Der Variablen `aktion` muss entweder der String `read` oder `write` zugewiesen werden. `read`, um die TXT-Datei, deren Name mit übergeben wird, auszulesen. `write`, um in die jeweilige Datei zu schreiben.

Wenn das CGI aufgerufen wird, werden dort durch die Funktionen `CGIreadData` und `CGIprepData` die übermittelten Daten ausgewertet und für das CGI verfügbar gemacht.

```
$Data = &CGIreadData;
%Data = &CGIprepData ($Data);
```

Die beiden Funktionen wurden in dem Sendmail-Beispiel → 107 „Versenden einer E-Mail“ erläutert.

Nachdem die übermittelten Daten dem CGI zu Verfügung stehen, wird mit einer If-Abfrage festgestellt, ob Daten gelesen oder geschrieben werden sollen.

```
if ( $Data{'aktion'} eq "write" )
{ &write_file; }
elsif ( $Data{'aktion'} eq "read" )
{ &read_file; }
else { print STDERR ("Es wurden keine Aktion empfangen\n"); }
```

Für das Überprüfen muss dann die Variable `aktion` übergeben werden. Falls diese keinen Wert haben sollte, wird eine Fehlermeldung angezeigt.

Wenn die übergebene Variable `aktion` den String `write` enthält, ruft die If-Abfrage die Funktion *write*`_file` auf.

```
sub write_file
{
open (FILE, ">>$datvrz"."$Data{'name'}.txt");
while ( ($temp, $wert) = each %Data )
{ print FILE "\u\&$temp=$wert\&\n";
$temp=0; }
print "&write=true&";
close FILE;
}
```

Diese Funktion öffnet das entsprechende TXT-File, schreibt alle Variablen hinein und schließt sie anschließend wieder. Nach Beendigung dieser Aktion übergibt dieser Aufruf noch die Variable `write` mit dem Wert true an Flash zurück.

Diese Funktion müssen Sie eventuell ändern. Durch den Befehl `open (FILE, ">>$datvrz"."$Data{'name'}.txt");` wird eine Datei geöffnet und die Daten des CGIs werden an die bisherigen Daten des TXT-Files angehängt, wie man es zum Beispiel bei einem Gästebuch braucht. Bei einem Counter hingegen muss man die bestehende TXT-Datei überschreiben; dann sieht der Befehl so aus:

```
open (FILE, ">$datvrz"."$Data{'name'}.txt");
```

Der Unterschied liegt in dem Zeichen >. Es bedeutet, dass die Datei überschrieben werden soll. zwei >-Zeichen bedeuten, dass Daten an die bestehende Datei angehängt werden sollen.

Wenn die Funktion `read_File` aufgerufen wird, öffnet diese ein File, liest die Daten heraus und übermittelt sie an Flash.

```
sub read_file
{
open (FILE, "<$datvrz"."$Data{'name'}.txt");
@file = <FILE>;
close FILE;
foreach $var (@file) { print $var; }
print "&eof=true&";
}
```

Damit man von Flash aus den Zeitpunkt abfragen kann, wann alle Daten empfangen wurden, übergibt das CGI als letzte Variable `eof` mit dem Wert true.

Wie schon bei dem CGI für das Versenden von Mails, müssen Sie dieses CGI abändern. Im Kopf ist die Adresse des Servers angegeben. Dort müssen Sie Ihren Server eintragen.

```
$datvrz = "/www/Server.de//cgi-bin/";
```

Mit diesem CGI kann man sehr viel Projekte verwirklichen, sei es, um einen Counter oder ein Gästebuch zu erstellen oder eine Rangliste für ein Spiel oder einen Chat.

Häufige Anwendungen wie das Lesen und Schreiben eines TXT-Files bei einem Chat beanspruchen den CGI-Server sehr stark. Dies ist auch ein Grund dafür, warum Sie sich vor dem Hochladen eines solchen Files bei Ihrem Webspace-Provider nach den Serverkapazitäten erkundigen sollten.

Zugriffszähler (Counter)

Auf vielen Webseiten zeigt ein Zähler die Anzahl der Besucher an. Hier wird erläutert, wie man Variablen an ein CGI schickt und empfängt und wie ein Counter aufgebaut sein muss.

Ordner Counter, Datei: counter.fla

▶ Funktionsprinzip

Der Zugriffszähler zählt die Anzahl der Seitenaufrufe. Wenn ein Besucher die Seite betritt, wird das Movie des Zugriffzählers einmal gestartet. Dieses Movie lädt zuerst den aktuellen Zählerstand aus einer Datei. Diese Datei wurde vom CGI erstellt und entspricht der Syntax von Flash. Diese Datei kann Flash also direkt einladen, wie schon im Kapitel *5* ➜ *82 Externe Dateien einlesen* beschrieben. Da es dabei aber vorkommen kann, dass eine Datei aus dem Cache geladen wird, benutzt man für das Auslesen der Datei ebenfalls ein CGI. Dies hat den Vorteil, dass man die TXT-Datei in dem cgi-bin-Verzeichnis erstellen kann, so dass Außenstehende auf diese Datei nicht zugreifen können.

Danach wird überprüft, ob die Datei komplett eingeladen wurde. Dies geschieht mit einer Variable Namens `eof`. Diese wird von dem CGI an die zu übermittelnden Daten als letzte Variable angehängt. Nach dem Einladen

wird die Anzahl der bisherigen Besucher um 1 erhöht und das CGI wieder aufgerufen, welches diesen Wert dann in die Datei auf dem Server schreibt.

▶ Allgemeiner Aufbau

Der komplette Counter befindet sich in einem Movie, dadurch ist er sehr universell einsetzbar. Theoretisch könnte man ohne große Probleme auch einen Smartclip daraus erstellen.

In dem Movieclip des Counters befindet sich ein weiterer Movieclip, der dazu dient, nur bestimmte Daten an das CGI zu schicken und nicht alle, die in diesem Movieclip benutzt wurden. Der Movieclip des Counters enthält folgendes `onClipEvent`:

Erstellen eines Smartclips
→ 92 Erstellen von Smartclips.

```
onClipEvent (load) {
    eof = false;
    daten.aktion = "read";
    daten.name = "counter";
    daten.gotoAndStop(2);
}
```

Nachdem das SWF eingeladen wurde, wird das `onClipEvent(load)` ausgeführt. `eof` wird gleich false gesetzt, um später abfragen zu können, ob dieses von dem CGI wieder `true` gesetzt wurde. Der Movieclip `daten` enthält die Variablen, die an das CGI übergeben werden müssen. Dies ist einmal die Variable `aktion` mit dem String *read* und zum anderen die Variable `name` mit dem Namen der TXT-Datei. Hier würde die TXT-Datei später auf dem Server counter.txt heißen. Der Befehl `daten.gotoAndStop(2);` lässt den Movieclip `daten` zu dem zweiten Frame springen, in dem folgendes ActionScript ausgeführt wird:

```
loadVariables("http://BlueBook.gnw.de/cgi-bin/counter.cgi",
"_root.counterm", "POST");
```

Das ActionScript übergibt die Variablen aus dem Movieclip `daten` an das CGI, die empfangenen Daten von dem CGI werden in dem Verzeichnis `_root.counterm` gespeichert.

In dem Moveiclip *Counterm* wird der erste Frame so lange abgespielt, bis `eof` von dem CGI true gesetzt wurde, da im zweiten Frame eine `gotoAndPlay (1);`-Anweisung sitzt.

```
if (eof eq "true") {
    counter = parseFloat(counter)+1;
    daten.name = "counter";
    daten.aktion = "write";
    daten.counter = counter;
    daten.gotoAndStop(3);
    gotoAndStop (3);
}
```

Wenn die If-Abfrage erfüllt ist, also das CGI alle Daten aus der TXT-Datei ausgelesen und an Flash geschickt hat, wird der Counter um 1 erhöht und wieder an das CGI zurückgeschickt. Da von dem CGI nur Strings übergeben werden, muss man den Wert des Counters zuerst mit `parseFloat` ermitteln, um ihn anschließend erhöhen zu können. Danach werden die Variablen `aktion` und `counter` in dem Movieclip `Daten` gesetzt. `Name` braucht nicht neu übergeben zu werden, da sich diese Angaben noch dort befinden. Die Variable `aktion` enthält nun den String *write*, wodurch das CGI die übermittelten Daten in die Textdatei schreibt. Durch den Befehl `gotoAndStop (3);` springt das aktuelle Bild des Movies *counterm* zu dem dritten Frame mit der Anzeige des Counters. Da das CGI noch aufgerufen werden muss, geschieht dies mit `daten.gotoAndStop(3);`. Dies bewirkt, dass das Frame mit dem folgenden ActionScript in dem Movie *Daten* aufgerufen wird.

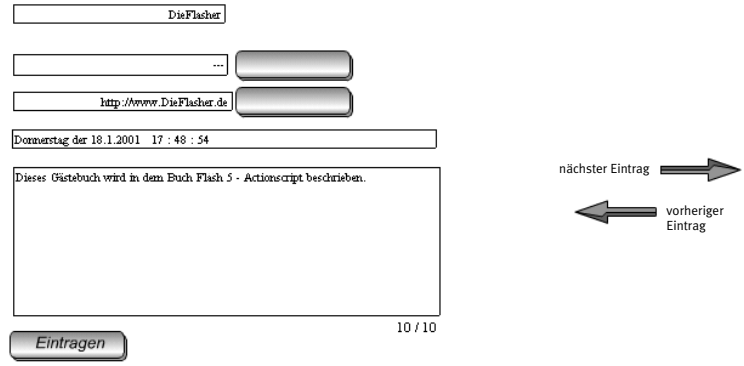
```
loadVariables ("http://BlueBook.gnw.de/cgi-bin/counter.cgi",
"", "POST");
```

Da keine Daten mehr von dem CGI unter Flash benötigt werden, wird auch kein Zielpfad für diese mitangegeben. Das CGI würde die Variablen `name`, `aktion` und `counter`, welche jetzt um 1 erhöht wurden, in die Textdatei schreiben. Das Movie zeigt den aktuellen Zählerstand an, so dass keine weiteren Aktionen mehr notwendig sind.

Gästebuch

Ordner Gastbuch, Datei: gb.fla

Ein Gästebuch gibt dem Besucher die Möglichkeit, einen für alle anderen User lesbaren Kommentar zu schreiben.



▶ Funktionsprinzip

Sobald das Gästebuch aufgerufen wird, werden dem Besucher alle bisherigen Einträge angezeigt. Dieser kann mittels zwei Buttons zwischen den älteren Einträgen hin und her wechseln. Durch einen weiteren Button hat der Besucher die Möglichkeit, in das Gästebuch zu schreiben. Danach kommt er wieder zur Anfangsübersicht zurück.

▶ **Allgemeiner Aufbau**

Das komplette Gästebuch befindet sich in einem Movie mit dem Instanznamen *gb*. Diesen Instanzen sind als `onClipEvent(load)` ähnliche Befehle zugeordnet wie dem Counter, da auch hier zu Beginn alle Daten aus der TXT-Datei gelesen werden müssen.

```
onClipEvent (load) {
    zaehler = false;
    eof = false;
    daten.aktion = "read";
    daten.name = "gb1";
    daten.gotoAndStop(2);
}
```

Die Variable `zaehler` wurde hier noch hinzugefügt. Diese Variable wird nach dem Eintrag ins Gästebuch gleich true gesetzt, um einen Doppeleintrag zu verhindern.

In dem Movie *gb* befindet sich ein Movieclip *daten*, in dem das CGI aufgerufen wird, um eine TXT-Datei auszulesen. In dem Movie *gb* gibt es zudem noch den Movieclip *senden*; dieser beinhaltet den Aufruf für das CGI, in eine Datei zu schreiben. Die Anweisung an das CGI, in eine Datei zu schreiben oder aus einer Datei zu lesen, ist in beiden Fällen an sich identisch (`loadVariables`). Der Unterschied liegt in den Variablen, die sich im jeweiligen Movieclip befinden. Damit keine überflüssigen Variablen mitgesendet bzw. in das Textfile geschrieben werden, haben wir dies in zwei getrennten Movies erstellt.

Nachdem das `onClipEvent(load)` ausgeführt wurde, wiederholt das Movie *gb* die folgende If-Abfrage im ersten Frame solange, bis diese wahr wird. Dieser Frameloop läuft solange, bis alle Variablen aus dem TXT-File an Flash gesendet wurden.

```
if (eof eq "true") {
    for (j=1; this["gesamtdate"+j] ne ""; j++) {
    }
    anzahl = j-1;
    i = anzahl;
    username = this["username"+i];
    email = this["email"+i];
    url = this["url"+i];
    text = this["text"+i];
    gesamtdate = this["gesamtdate"+i];
    gotoAndStop (3);
    vonbis = i+" / "+anzahl;
}
```

Da im Gästebuch im Gegensatz zum Counter das Textfile nicht komplett überschrieben wird, sondern der neue Eintrag nur angehängt wird, existiert

keine Variable für die Anzahl der Einträge – ansonsten müsste man diese Variable in dem TXT-File überschreiben. Man hätte die alten Einträge zwar unten noch einmal in das Textfeld mit hinhein schreiben können – die zuletzt ausgelesenen Werte sind nämlich die gültigen, falls zweimal die gleiche Variable vorkommen sollte. Diese Lösung ist aber weniger elegant und das TXT-File würde unnötig groß. Deshalb existiert keine Variable `anzahl`, sondern die Anzahl der Einträge wird durch die folgende for-Schleife ermittelt.

```
for (j=1; this["gesamtdate"+j] ne ""; j++) {
}
anzahl = j-1;
```

Die for-Schleife läuft solange, bis sie keinen Textstring in der Variable `gesamtdate` mehr findet. Dies bedeutet, dass dort noch kein Eintrag erfolgt ist, da das Datum immer mit übertragen wird. Wenn Sie von der Anzahl der Schleifendurchläufe 1 abziehen, erhalten Sie die Zahl der Einträge.

Die Einträge werden durchnummeriert, d.h. der erste Eintrag benutzt die Variablen `name1`, `email1`, `url1` ..., der zweite `name2`, `email2`, `url2` ... usw.

`i` wird gleich `anzahl` gesetzt, damit der letzte Eintrag abgebildet wird. Die Textfelder auf dem Bildschirm heißen `name`, `email`, `url` ..., diesen weist man nun den entsprechend letzten Eintrag zu.

```
i = anzahl;
username = this["username"+i];
email = this["email"+i];
url = this["url"+i];
text = this["text"+i];
    gesamtdate = this["gesamtdate"+i];
```

Damit das Gästebuch keinen Schriftzug „Loading“ mehr anzeigt, der sich in den ersten zwei Frames befindet, muss zum dritten Frame gesprungen werden, in dem sich das eigentliche Gästebuch befindet.

```
gotoAndStop (3);
```

In dem Textfeld `vonbis` werden die Nummer des gelesenen Eintrags und die Gesamtzahl der Einträge angezeigt:

```
vonbis = i+" / "+anzahl;
```

Um zwischen den verschiedenen Einträgen wechseln zu können, benutzt man zwei Buttons, einmal für den nächsthöheren Eintrag und einmal für den vorherigen. Der Button für den nächsten Eintrag sieht wie folgt aus.

```
on (release) {
    if (i != anzahl) {
        i++;
        vonbis = i+" / "+anzahl;
        gesamtdate = this["gesamtdate"+i];
```

```
            username = this["username"+i];
            email = this["email"+i];
            url = this["url"+i];
            text = this["text"+i];
        }
    }
```

Wenn i nicht so groß wie `anzahl` ist, da `anzahl` der Nummer des letztens Eintrags entspricht, wird i um 1 erhöht. Anschließend werden den Textfeldern die neuen Inhalte zugewiesen.

Der Schalter für den vorherigen Eintrag sieht fast genauso aus. Nur dass bei diesem nicht i erhöht, sondern verringert wird.

```
on (release) {
    if (i>1) {
        i--;
        vonbis = i+" / "+anzahl;
        gesamtdate = this["gesamtdate"+i];
        username = this["username"+i];
        email = this["email"+i];
        url = this["url"+i];
        text = this["text"+i];
    }
}
```

`i` darf nicht kleiner als 1 werden, da z. B. `name1` der Name des ersten Eintrags ist. Wenn `i` gleich 0 gesetzt wird, würde man auf die Variable `name0` zugreifen, die gar nicht existiert.

Hier werden die Einträge seitenweise angezeigt. Mit Hilfe der Texteffekte (→ 68) können Sie die Einträge wie bei den meisten Gästebüchern auch untereinander anzeigen.

Ein weiterer Button wird benötigt, um einen Eintrag in das Gästebuch schreiben zu können.

```
on (release) {
    if (!zaehler) {
        gotoAndStop (4);
        username = "Ihr Name";
        email = "Ihre E-Mail-Adresse";
        url = "http://www.";
        text = "Ihr Kommentar";
    } else {
        gotoAndStop (3);
        text = "Du hast dich eben schon in diesem Gästebuch
        eingetragen !";
        vonbis = "-- / "+anzahl;
        username = "Systemadmin";
```

```
            email = "---";
            url = "---";
            gesamtdate = "---";
        }
    }
```

Dieser überprüft zuerst, ob sich der Besucher zum zweiten Mal eintragen möchte, und gibt dann die Meldung `text = "Du hast dich eben schon in diesem Gästebuch eingetragen !";` aus oder lässt den Movie *gb* in den nächsten Frame springen, in dem die Eingabefelder liegen. Den Eingabefeldern werden Strings zugewiesen, damit der User weiß, wofür welches Eingabefeld dient.

```
username = "Ihr Name";
email = "Ihre E-Mail-Adresse";
url = "http://www.";
text = "Ihr Kommentar";
```

In Frame 4 liegen zwei Buttons, einer, um wieder zurückzugehen und nichts ins Gästebuch einzutragen, und ein zweiter zum Eintrag.

Der Abbruchbutton verweist mit einem `GotoandStop`-Befehl auf Frame 3, in dem der Inhalt des Gästebuches angezeigt wird.

Der Eintragsbutton enthält folgendes ActionScript:

```
on (release) {
    zaehler = true;
    play ();
    eof = false;
    daten.aktion = "read";
    daten.name = "gb1";
    daten.gotoAndStop(2);
}
```

Im unteren Teil findet sich der Aufruf zum Einlesen der TXT-Datei. Dieser Eintrag ist notwendig, damit gleichzeitige Einträge sich nicht überschreiben. Deshalb wird vorher noch einmal das TXT-File auf neue Einträge überprüft. Des Weiteren wird die Variable `zaehler` auf `true` gesetzt, damit der Besucher sich anschließend nicht noch einmal in das Gästebuch eintragen kann. Das Movie *gb* wird ab Frame 4 abgespielt, in Frame 5 befindet sich dann ein ActionScript, welches anhand der Variable `eof` überprüft, ob das TXT-File eingeladen wurde.

```
if (eof eq "true") {
    for (j=1; this["gesamtdate"+j] ne ""; j++) {
    }
    anzahl = j-1;
    i = anzahl+1;
    vonbis=i+" / "+anzahl;
    senden["username"+i] = username;
```

```
        senden["email"+i] = email;
        senden["url"+i] = url;
        senden["text"+i] = text;
        senden["gesamtdate"+i]=gesamtdate;
        write = false;
        senden.aktion = "write";
        senden.name = "gb1";
        senden.gotoAndStop(3);
        gotoAndPlay (6);
    } else {
        prevFrame ();
        play ();
    }
```

Falls das TXT-File noch nicht komplett eingeladen sein sollte, spielt das Movie vom vorherigen Frame ab und bildet somit auch wieder ein Frameloop.

Wenn das TXT-File eingeladen wurde, also die neueste Version des Gästebuches vorliegt, erfolgt der Aufruf des CGIs für das Schreiben. Zuerst wird wieder die Anzahl der Einträge aktualisiert:.

```
for (j=1; this["gesamtdate"+j] ne ""; j++) {
}
anzahl = j-1;
i = anzahl+1;
```

Danach werden die Variablen, die in das TXT-File geschrieben werden sollen, in das Movie *senden* kopiert. Dazu müssen auch die Variablen gesetzt werden, die das CGI benötigt, um zu wissen, dass Daten geschrieben und in welches File sie geschrieben werden sollen.

Durch `senden.gotoAndStop(3);` springt das Movie *senden* zu Frame und führt dort folgendes ActionScript aus. Wir haben die `read`- und `send`-Aufrufe für das CGI absichtlicht in verschiedene Frames gelegt, damit man sie nicht verwechseln kann.

Der Aufruf des CGIs in dem Movie *senden* im dritten Frame sieht wie folgt aus:

```
loadVariables ("http://BlueBook.gnw.de/cgi-bin/gb.cgi",
"_root.gb", "POST");
gotoAndStop (1);
```

Dort werden alle Variablen, die in dem Movieclip *senden* liegen, an das CGI übermittelt. Die Daten, die von dem CGI zurückgesendet werden, speichert das CGI in dem Movieclip `_root.gb`.

Der Movieclip *gb* springt, wenn der Befehl für das Senden der Variablen erfolgt, zu Frame 6.

```
gotoAndPlay (6);
```

Dort befindet sich auch ein Frameloop; in Frame 10 befindet sich die dazugehörige Abfrage:

```
if (write eq "true") {
    gotoAndPlay (1);
    eof = false;
    daten.aktion = "read";
    daten.name = "gb1";
    daten.gotoAndStop(2);
} else {
    gotoAndPlay (6);
}
```

Dieses Frameloop überprüft, ob das CGI die Daten geschrieben hat. Wenn dies zutrifft, setzt das CGI die Variable `write=true` und die Befehle in der If-Abfrage würden ausgeführt.

Diese Befehle laden per CGI noch einmal das TXT-File neu ein, damit der Besucher seinen neuen Eintrag auch direkt in dem Gästebuch sieht. Es wird zu Frame 1 gesprungen, da sich dort der Frameloop mit der Abfrage, ob das Einlesen beendet wurde, befindet. Das Gästebuch wird neu gestartet, der einzige Unterschied liegt nun darin, dass die Variable `zaehler=true` ist und sich der Besucher nicht noch einmal eintragen kann.

Hier nur eine kurze Beschreibung der Variablenübergabe zwischen PHP und Flash.

Da wir in diesem Buch ausführlicher auf das Zusammenspiel von Flash und CGI eingehen, werden wir hier keine Beispiele mehr anführen. Mit den gezeigten CGIs lässt sich alles so weit realisieren.

Falls Sie jedoch lieber mit PHP arbeiten als mit CGI, folgt eine kurze Einweisung, wie die Variablen von Flash an PHP und umgekehrt übergeben werden können:

Genau wie beim CGI wird das PHP mit `loadVariables ()` aufgerufen und die Variablen mit POST oder GET übergeben. Der `loadVariables`-Befehl übergibt alle Variablen, die in dem gleichen Movie wie der Befehlsaufruf liegen, in der Reihenfolge ihrer Initialisierung an das PHP. In dem PHP können die Variablen dann mit `$variablenname1`, `$variablenname2` usw. angesprochen werden.

Um Variablen aus dem PHP wieder an Flash zurückzugeben, muss man dies mit `echo"variablenname1=$wert1\n";`
bzw. bei mehreren Variablen mit

```
echo"variablenname1=$wert1&variablenname2=$wert\n";
```

an Flash übermitteln. Danach stehen die Variablen in Flash in dem Movie, das bei `loadVariables ()` als Ziel angegeben war, zur Verfügung.

6 Spieleprogrammierung

Endlich – mit der fünften Version – bietet Flash zusätzlich zu der Vielzahl seiner anderen Möglichkeiten einen vernünftigen Ansatz zur Spieleprogrammierung.

Wer jedoch an den Entwurf einer 3D-Engine unter Flash denkt, der muss seine Erwartungen zügeln. Ebenso stellen schnelle großflächige Tweenings ein Problem dar.

Selbst leistungsfähige PCs kann Flash überfordern. Zum einen durch das Duplizieren von mehreren Movies, die ein Tweening enthalten, oder durch 2 bis 3 Alphatweenings im Vollbildmodus. Dabei kann die Framezahl sogar unter 5 Fps sinken. Seien Sie sich dieser Grenzen, die Flash setzt, bewusst.

Flash besitzt eine Menge neuer Funktionen; besonders in Bezug auf die Spielprogrammierung sind die neuen Funktionen wie `HideMouse` oder `SwapDepth` sehr nützlich. Die neue Abfragemöglichkeit der Tastatur gewährleistet eine bessere Steuerung des Spiels über die Tastatur. (Bei Flash 4 kam es zu Problemen, wenn mehr als zwei Tasten gedruckt wurden oder man abfragen wollte, ob eine Taste wieder losgelassen wurde.)

Spiele ohne rasche Bildwechsel sind problemlos zu erstellen: also Denk-, Brett- oder Strategiespiele. Selbst Jump and Runs oder die beliebten Acardeklassiker erhalten durch Flash einen ungeahnten Aufschwung.

Die Euphorie zu Beginn der Entwicklung eines neuen Spiels ist groß: Beachten Sie aber: Je umfangreicher ein Spiel wird, desto unübersichtlicher wird es. Entwickeln Sie deshalb zunächst ein Konzept, wie Sie das Spiel am besten unter Flash verwirklichen können. Hilfreich ist genügend Erfahrung im Umgang mit Flash; denn erst, wenn Sie alle Funktionen unter Flash kennen, finden Sie die einfachste Lösung zur Verwirklichung Ihrer Visionen.

Deshalb beschreiben wir im Folgenden häufig vorkommende Situationen in der Spielprogrammierung und wie man diese am geschicktesten verwirklicht. Bei den gängigsten Spiele gibt es zumindest in Hinsicht des Funktionsumfangs von Flash, vier unterscheidbare Merkmale:

- Duplizieren/Kopieren von „Objekten“
- Eigenschaften von „Objekten“ wie Beschleunigung oder Gravitation
- Arbeiten mit mehreren Objekten (Abfragen, Schleifen, Strukturen, Funktionen, objektorientierte Programmierung)
- Errechnen von 3D-Objekten unter Flash

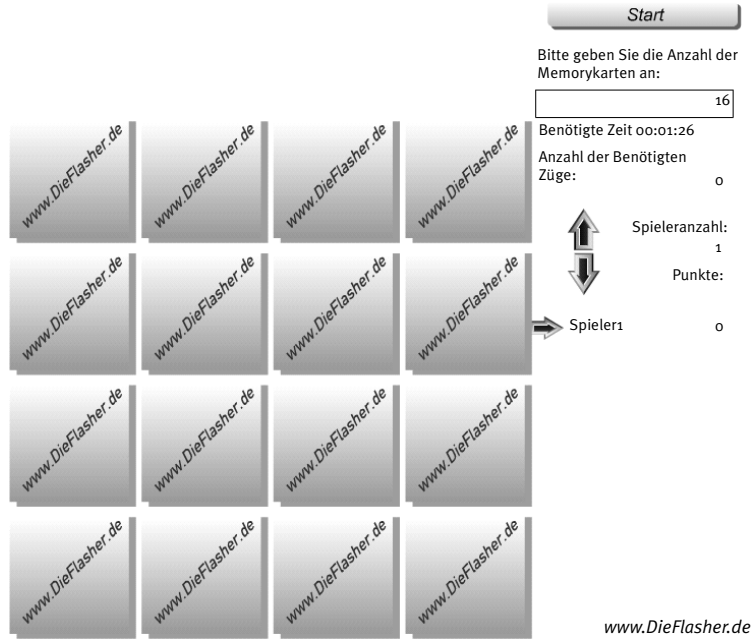
Machen Sie sich vor der Entwicklung eines neuen Spiels Gedanken über das Spielprinzip, den Aufbau und seine Verwirklichung in Flash. Erst wenn man eine grobe Vorstellung von der Realisierung des kompletten Spiels hat, sollte man anfangen, das Spiel zu entwickeln.

Beginnen Sie am besten mit einer kleinen, aber für das Spiel grundlegenden Funktion: z.B. bei Memory, wie die Karten auf dem Bildschirm erscheinen oder wie eine Karte aufgebaut ist; bei einem Autorennen die Fahrzeugsteuerung und die Beschränkung der Fahrbahn. Widmen Sie sich einem neuen Problem erst, wenn Sie das vorherige gelöst haben.

6.1 Memory

Ordner Memory, Datei: memory.fla

Allgemeine Überlegungen



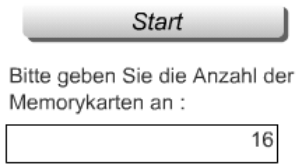
Bei Memory geht es darum, aus mehreren verdeckt gelegten Karten die zwei gleichen herauszufinden. Da alle Karten prinzipiell gleich aufgebaut sind, kommt man schnell auf die Idee, diese mit `duplicateMovieClip()` zu erstellen (möglich wäre auch `attachMovie()`), da alle Karten die gleichen Aktionen bei einem Klick ausführen sollen. Die Aktion einer Karte ist also folgende: Karte aufdecken, mit der anderen Karte, falls diese schon aufgedeckt ist, vergleichen und auswerten. Da jede Karte eine Vorder- und eine Rückseite besitzt, sollte man diese in einem Movie erstellen. Dies ist sowieso nötig, da man sonst auch nicht den Befehl `duplicateMovieClip()` anwenden könnte. Da die Karte als Movie angelegt wurde, kann man auch

gleich die passende Rückseite erzeugen. Wenn man nun Bilder einfügen möchte, fügt man diese in die hinteren Frames der Karte ein und lässt das Movie zu den Frames entsprechend dem zugeordneten Wert springen. Dieses Beispiel haben wir allerdings mit Zahlen und Farben gelöst, da wir sonst ziemlich viele Bilder hätten einfügen müssen. Nun erfolgt die Überlegung, wann und wie oft die Karten dupliziert werden sollen. Wenn man nun die Karten schon dupliziert, kann man die Entscheidung über die Anzahl der Karten gleich dem Besucher überlassen und dafür ein Eingabefeld erzeugen. Der Befehl `duplicateMovieClip()` wird also von dem Besucher durch einen Startbutton ausgelöst. Gleichzeitig mit `duplicateMovieClip()` sollte man den Karten auch einen Wert zuweisen, da man für das Duplizieren der Movies sowieso schon eine entsprechende Schleife erzeugen muss. Wenn man Bilder eingefügt hätte, würde man auch Werte zuweisen müssen, um über den entsprechenden Wert jeweils zum Frame mit dem passenden Bild springen zu können. Sie können dies durch den Befehl `gotoandplay(wert)` einbinden.

Da alle Werte aber genau zweimal vorkommen müssen und auch zufällig verteilt sein sollten, generiert man am besten ein Array, das alle zur Verfügung stehenden Zahlen enthält. Wenn man der ersten Karte eine Zahl zuweist, löscht man diese Zahl aus dem Array – dafür gibt es in Flash 5 die Operation `splice()`. Der Befehl `splice()` wird weiter unten noch genau beschrieben. Da das Array genau alle vorkommenden Zahlen enthält und man diese mit `random()` herausliest, besitzt jede kopierte Karte einen Zufallswert, der bei allen Karten insgesamt genau zweimal vorkommt. Öfters vorkommende Operationen verwirklicht man am besten in einer Schleife.

Der Anfang

Beginnen Sie mit der Entwicklung des Spiels, wenn Sie sich über die Grundprinzipien im Klaren sind und diese schriftlich festgehalten haben. Die meisten Spiele fangen mit einem Startbutton an, so auch dieses Memoryspiel. Man erstellt einen Button, um die Karten zu duplizieren. Da die Anzahl der Memorykarten dem User überlassen wird, erzeugt man neben dem Button ein Textfeld, um die Anzahl der Memorykarten einzugeben. Um die Eingabe zu beschränken, legt man die maximal einzugebende Zahl auf drei Stellen fest *Max.ZCH.: 3*. Der Button sollte sich allerdings in einem Movieclip befinden, damit der Button nicht aktiv dargestellt wird, falls das Eingabefeld keine oder eine falsche Eingabe enthält.



Das Movie soll im ersten Frame lediglich ein Bild des Buttons enthalten (nur die Grafik, unter Umständen grau unterlegt) und erst im zweiten Frame den aktiven Button.

Damit der richtige Button angezeigt wird, überprüft man kurz das Textfeld auf eine

Eingabe, die größer als 0 ist, da man einen Wert für die Berechnung benötigt.

Da diese Überprüfung dauerhaft stattfinden soll, geht dies am besten mit `onclipevent(enterFrame)`. Dieses erstellt man im Movie des Schalters. Das hat den Vorteil, dass man anschließend keine Pfadangaben mehr verwenden muss, wenn man den Button zu einem andere Frame springen lassen möchte. Das Textfeld, in dem der User die Anzahl eingeben kann, besitzt den Namen *Anzahl*.

```
onClipEvent (enterFrame) {
    if (_root.anzahl>0) {
        gotoAndStop (2);
    } else {
        gotoAndStop (1);
    }
}
```

Hier wird die Eingabe in dem Textfeld *Anzahl* überprüft, und sobald eine Zahl größer als 0 eingegeben wird, springt der Movieclip zu dem aktiven Button. Falls die Eingabe kleiner als 0 ist, springt der Movieclip wieder zurück zu der grauen Button-Grafik.

So ist es immer nur dann möglich, den Startbutton zu drücken, wenn sich in dem Textfeld *Anzahl* eine Zahl größer als 0 befindet.

Das Duplizieren der Memorykarten

Es folgt jetzt der schwierigere Teil. Durch Drücken des Startbuttons muss der Movieclip *Karte,* entsprechend der eingegebenen Anzahl, dupliziert werden. Die duplizierten Karten müssen auch auf dem Bildschirm – entsprechend ihrer Anzahl – ausgerichtet werden.

Die Anzahl muss einem geraden Wert entsprechen, da eine ungerade Anzahl an Karten nicht möglich ist. Dies geschieht mit folgendem ActionScript:

```
if (_root.anzahl%2 == 1) {
    _root.anzahl++;
}
```

Diese Operation % wird als `modulo` bezeichnet und teilt eine Zahl durch ihre nachfolgende und gibt als Ergebnis den verbleibenden Rest heraus: Bei Eingabe der *Anzahl* 9, ergibt dies 1, da 9 : 2 = 4 Rest 1 ist. Diese If-Abfrage veranlasst ein `True` und dadurch wird `Anzahl` durch den Befehl `_root.anzahl++;` um eins erhöht, also aufgerundet. Wenn man an dieser Stelle `_root.anzahl--;` einsetzt, rundet man ab.

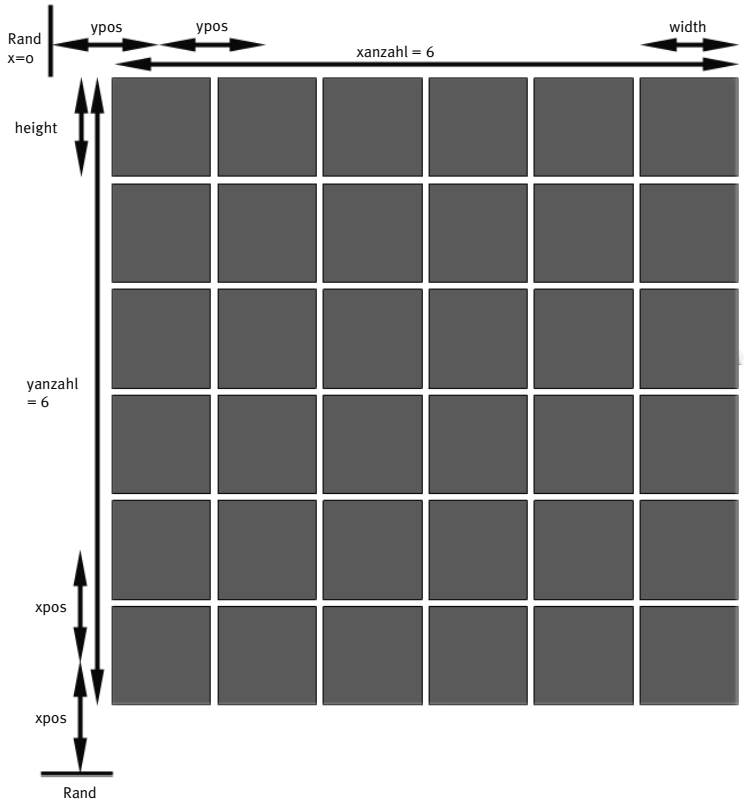
Nachdem Sie eine gerade *Anzahl* erstellt haben, sorgen Sie dafür, dass diese *Anzahl* größer als 3 ist, da ein Spiel mit 2 Karten keinen Sinn macht.

```
if (_root.anzahl<4) {
```

```
        _root.anzahl = 4;
    }
```

Damit wird eine eingegebene *Anzahl*, die kleiner als 4 ist, automatisch auf 4 erhöht.

Nachdem Sie für die geeigneten Startbedingungen gesorgt haben, entwickeln Sie eine Formel, die den Abstand zwischen den Kartenmittelpunkten sowie die Kartengröße errechnet, da diese je nach der Anzahl variiert.



Da man versucht, die Karten quadratisch auszulegen, kann man die Wurzel aus der Anzahl bilden, diese runden und daraus auf die Anzahl der untereinanderliegenden Karten schließen (`yanzahl`):

```
yanzahl = int(Math.sqrt(_root.anzahl));
```

Da man hier mit `int` abgerundet hat, muss man bei der Berechnung der `xanzahl` wieder aufrunden, da beim Wiederabrunden ein zu kleines Feld erzeugt würde. Das Aufrunden geschieht durch Addition der Zahl 0,9999, da hier, sobald eine Zahl nur geringfügig größer als 1 ist, aufgerundet werden muss.

```
xanzahl = int((_root.anzahl/yanzahl)+0.9999);
```

Wenn man eine Zahl mit int runden möchte, also Zahlen größer als X,5 aufrunden und Zahlen unter X,5 abrunden, müsste man 0,5 zu der jeweiligen Zahl addieren.

Auf- und Abrunden
→ *39 Mailfunktionen.*

In Flash 5 gibt es allerdings eine Funktion, die einfach auf die nächst höhere ganze Zahl aufrundet.

```
xanzahl = Math.ceil(_root.anzahl/yanzahl);
```

Durch die Berechungen weiß man nun, wie viele Karten untereinander und wie viele Karten nebeneinander gesetzt werden müssen. Jetzt errechnet man noch Abstände und Größe der einzelnen Karten.

```
xpos = 600/(xanzahl+1);
width = xpos-5;
ypos = 600/(yanzahl+1);
height = ypos-5;
```

Ein Rand für die Ausgabe in der Größe von 200 Pixel ist vorteilhaft, da die Spielfeldgröße dann 600x600, also quadratisch, ist.

Der Wert 600 steht für die Spielfeldgröße. Die Ausgabegröße des SWFs beträgt 800 x 600 Pixel. Da für die Anzeige am Rand noch ein Platz von 200 Pixel gelassen werden muss, ist die maximale Höhe 600 und die maximale Breite 800 – 200, also 600.

Um den Abstand zwischen den Karten zu errechnen, teilt man den zu Verfügung stehenden Platz durch Anzahl + 1. Man muss den Teiler um 1 erhöhen, da man den Abstand zwischen den Mittelpunkten errechnen möchte. Wenn man von vier nebeneinander liegenden Rechtecken die Mittelpunkte errechnen möchte, muss man die Breite der Rechtecke durch 5 teilen, deshalb Anzahl + 1. Da die Rechtecke alle gleich groß sind, kann man, um die Breite zu ermitteln, den X-Abstand zwischen den errechneten Mittelpunkten als Breite verwenden. Dadurch würden die Karten aber abstandslos aneinander liegen. Um dies zu verhindern, verringert man die Breite um 5 Pixel pro Karte – 2,5 Pixel auf jeder Seite, so dass der Abstand zwischen zwei Karten genau 2 x 2,5 = 5 Pixel beträgt.

Nach dem Abschluss aller Berechnungen kann man die Karten duplizieren und diesen ihre entsprechenden Positionen zuweisen. Hierbei benutzt man wieder eine Schleife, da ein und dieselben Befehle innerhalb eines Frames öfters ausgeführt werden müssen.

Die Befehle müssen genau so oft ausgeführt werden wie die gewünschte Anzahl der Karten. Deshalb lässt man einen Index `i` hochzählen, bis dieser genauso groß bzw. nicht mehr kleiner als `Anzahl` ist.

```
for (i=0; i<_root.anzahl; i++) {
```

In der for-Schleife befindet sich der Befehl `duplicateMovieClip`.

```
duplicateMovieClip (_root.karte, "karte"+i, i);
```

Hier wird der Movieclip *karte* kopiert. Der neue Movieclip hat den Namen *„karte“+i* . Die erste kopierte Karte heißt dann *Karte1*, die zweite *Karte2* usw. Dies kommt daher, weil sich `i` bei jedem neuen Befehlsaufruf um 1 erhöht und die Schleife bei 1 startet. Da man jeder Karte eine Tiefe zuordnen muss, dieses für das Spiel aber irrelevant ist, setzt man die Tiefe erst einmal bei jeder Karte entsprechend i, damit nicht zwei Karten die gleiche Tiefe bekommen, da sonst die letztere die zuerst Eingefügte ersetzen würde.

Nach dem Duplizieren der Karten muss man diesen noch Werte zuweisen. Da man für jede Karten einen Werte haben muss (wobei allerdings zwei Karten immer die gleichen Werte besitzen), braucht man genau [Anzahl/2 = Werte] x 2: genau die Hälfte von der Anzahl, da man nur die Hälfte an Werten benötigt, da alle Zahlen doppelt vorkommen. Diese Zahlen speichert man am besten in einem Array, um dadurch die gebrauchten Zahlen aus diesem Array löschen zu können, so dass keine Zahl bzw. kein Feld doppelt aus dem Array ausgelesen wird.

```
zahlen = new Array();
    for (i=0; i<(_root.anzahl/2); i++) {
        zahlen[i] = i+1;
        zahlen[i+_root.anzahl/2] = i+1;
    }
```

Zuerst wird wieder ein Array *Zahlen* erzeugt, das mit Zahlen beschrieben wird. Das erste Feld des Arrays `zahlen[0]` wird gleich i + 1 = 0 + 1, also auf 1 gesetzt, genauso wie das mittlere Feld `zahlen[0+_root.anzahl/2]`. Beim zweiten Durchlauf wird das zweite Feld des Arrays `zahlen[1]` auf 1 + 1, also auf 2 gesetzt, und das mittlere Feld + 1 `zahlen[1+_root.anzahl/2]` wird auch auf `1 + 1 = 2` gesetzt. Dadurch besitzt das Array immer zweimal die gleiche Zahl, genauso wie wir es für unser Memoryspiel brauchen.

Aus diesem Array liest man nun zufällig eine Zahl heraus und weist diese der duplizierten Karte zu. Das zufällige Auslesen einer Zahl geschieht mit der Funktion `random()`. Als Argument in der Klammer geben Sie die maximale Größe der zu ermittelnden Zufallszahl ein.

```
zufallszahl = random(_root.anzahl-i);
```

Es wird also eine Zufallszahl über die Anzahl der Karten ermittelt, z.B. bei 64 Karten eine Zahl zwischen 0 und 63. Die Zufallszahl dient dazu, ein zufälliges Feld des Arrays anzusprechen. Angenommen die Zufallszahl wäre 4, dann wird der Inhalt des vierten Feldes des Arrays *Zahlen* ausgelesen und der gerade duplizierten Karte zugewiesen. Die Zuweisung geschieht, indem man der Variablen *Kartennr* in dem Movieclip *karte* den Wert aus dem Array zuweist.

```
_root["karte"+i].kartennr = zahlen[zufallszahl];
```

Wenn diese Zahl übergeben wurde, wird das Feld des Array, aus dem die Zahl stammt, mit dem Befehl `Splice(Zahl des zu entfernenden Feldes)` entfernt.

```
zahlen.splice(zufallszahl, 1);
```

Bei einem Spiel mit 12 Memorykarten sieht das Array bei i = 0 folgendermaßen aus:

Array[0]	1
Array[1]	2
Array[2]	3
Array[3]	4
Array[4]	5
Array[5]	6
Array[6]	1
Array[7]	2
Array[8]	3
Array[9]	4
Array[10]	5
Array[11]	6

Nachdem die vierte Zahl bei *zufallszahl* = 4 mit `Splice` aus dem Array entfernt wurde, enthält das Array noch folgende Werte:

Array[0]	1
Array[1]	2
Array[2]	3
Array[3]	5
Array[4]	6
Array[5]	1
Array[6]	2
Array[7]	3
Array[8]	4
Array[9]	5
Array[10]	6

Dadurch lässt sich dann auch erklären, warum die Zufallszahl, die mit `random()` gebildet wird, bei jedem Schleifendurchlauf um 1 kleiner werden muss. Dies muss so sein, da auch dass Array bei jedem Schleifendurchlauf um 1 kleiner wird (`random(_root.anzahl-i);`). `i` wird bei jedem Schleifendurchlauf um 1 erhöht und damit wird der Wert, aus dem eine Zufallszahl ermittelt wird, jeweils um 1 kleiner.

Nachdem man der duplizierten Karte einen Wert zugewiesen hat, muss man diese an der richtigen Stelle am Bildschirm abbilden: Dafür errechnen Sie die X- und Y-Koordinaten sowie die Höhe und Breite der jeweiligen Karte und weisen Sie diesen Werte der jeweils duplizierten Karte zu.

Vor der for-Schleife wurden die Werte für die einzelnen Karten ermittelt.

xpos	X-Abstand zwischen den Mittelpunkten der Karten
ypos	Y-Abstand zwischen den Mittelpunkten der Karten
width	Breite der Karten
height	Höhe der Karten
yanzahl	Anzahl der Zeilen
xanzahl	Anzahl der Spalten

Zuerst kommt man auf die Idee, dass dies wie folgt aussehen müsste:

```
_root["karte"+i]._x = xpos*(i+1);
_root["karte"+i]._width = width;
_root["karte"+i]._y = ypos;
_root["karte"+i]._height = height;
```

Noch bevor man den Versuch startet, dürfte klar sein, dass dies nicht ausreicht. Zwar wird die entsprechende Karte immer um den entsprechenden Faktor nach rechts verschoben `xpos*(i+1)`, jedoch bleibt die Y-Position immer gleich. Deshalb müssen Sie noch Korrekturfaktoren in die Berechnung einfließen lassen. Diese zusätzlichen Faktoren ändern den Wert so, dass in dem Fall, dass die maximale Kartenanzahl in einer Zeile dargestellt wurde, `xpos` wieder von vorne zu zählen beginnt, und `ypos` sich um 1 erhöht, um in die nächste Zeile zu gelangen.

Deshalb müssen Sie noch folgende If-Abfrage miteinbinden.

```
if ((_root.anzahl/(i+xkorrektur)) == _root.anzahl/xanzahl) {
    xkorrektur -= xanzahl;
    ykorrektur += 1;
}
```

Diese Abfrage wird immer wahr, wenn das Zeilenende erreicht wurde und die nächste Karte in einer neuen Zeile erscheinen muss. Die x- und y-Korrekturwerte müssen dann noch beim Setzen der X-Y-Positionen als Faktoren eingebunden werden.

```
_root["karte"+i]._x = xpos*(i+1+xkorrektur);
_root["karte"+i]._width = width;
_root["karte"+i]._y = ypos*ykorrektur;
_root["karte"+i]._height = height;
```

Nachdem nun die komplette Aktion für das Duplizieren der Karten erstellt wurde, kann man die gleich bleibenden Berechnungen aus der for-Schleife heraus ziehen, um die Berechnungszeit zu verkürzen. Leider bringt dies nicht viel, da Flash die meiste Zeit für das Duplizieren der Karten benötigt.

Da man mit dem Startbutton auch Restarten kann, um ein laufendes Spiel abzubrechen und ein neues zu starten, muss der Schalter alle Variablen wieder auf die Startwerte setzen. Dies geschieht bei `on(press)`, da sich in dem Release-Befehl die Schleife mit dem `duplicateMovieClip()` befin-

det. Dieser Vorgang ist sehr rechenaufwändig ist, so dass Flash die vorhandenen Rechtecke erst entfernt, wenn er Position und Größe der neuen Rechtecke errechnet hat. Dadurch bleiben die alten Rechtecke während der Berechnungszeit auf dem Bildschirm sichtbar. Dies ist ungünstig, da der Button im Hit-Bereich einen Schriftzug enthält, der wegen der alten Memorykarten unleserlich dargestellt wird.

Dies zeigt auch, dass der Press-Befehl zuerst komplett abgearbeitet und dann erst der Release-Befehl ausgeführt wird. Die Grafik des Buttons Hit ist während der gesamten Rechenzeit bei `on(release)` sichtbar. Diese Eigenschaft haben wir hier benutzt, um dem User mitzuteilen, dass er sich einen Moment gedulden muss. Deshalb erscheint auch der Schriftzug während der Berechnungsphase auf dem Bildschirm.

Im Folgenden noch einmal das gesamte ActionScript des Buttons:

```
on (press) {
_root.timer.stopp=false;
_root.timer.sec=0;
_root.timer.min=0;
_root.timer.std=0;
      _root.verriegelt = false;
      _root.gewonnen.gotoAndStop(1);
      for (i=0; i<1000; i++) {
              removeMovieClip ("_root.karte"+i);
      }
      xkorrektur = 0;
      ykorrektur = 1;
      for (i=0; i<_root.spieleranzahl.spieleranzahl+1; i++) {
      _root.spieleranzahl.spielerfelder["spieler"+i].Punkte = 0;
      }
      _root.befehl.spieler = 1;
      _root.befehl.spielersichtbar();
      if (_root.anzahl<3) {
              _root.anzahl = 4;
      }
      if (_root.anzahl%2 == 1) {
              _root.anzahl++;
      }
      _root.anzahlderkarten = _root.anzahl;
}
on (release) {
      _root.gotoAndStop(2);
      zahlen = new Array();
      for (i=0; i<(_root.anzahl/2); i++) {
              zahlen[i] = i+1;
              zahlen[i+_root.anzahl/2] = i+1;
```

```
        }
        yanzahl = int(Math.sqrt(_root.anzahl));
        xanzahl = Math.ceil(_root.anzahl/yanzahl);
        xpos = 600/(xanzahl+1);
        width = xpos-5;
        ypos = 600/(yanzahl+1);
        height = ypos-5;
        anzahldx=_root.anzahl/xanzahl;
        for (i=0; i<_root.anzahl; i++) {
                duplicateMovieClip (_root.karte, "karte"+i, i);
                zufallszahl = random(_root.anzahl-i);
                _root["karte"+i].kartennr = zahlen[zufallszahl];
                zahlen.splice(zufallszahl, 1);
                if ((_root.anzahl/(i+xkorrektur)) == anzahldx) {
                        xkorrektur -= xanzahl;
                        ykorrektur += 1;
}
                _root["karte"+i]._x = xpos*(i+1+xkorrektur);
                _root["karte"+i]._width = width;
                _root["karte"+i]._y = ypos*ykorrektur;
                _root["karte"+i]._height = height;
        }
_root.timer.startwert=int(getTimer()/1000);
}
```

Der Aufbau einer Memorykarte

Die Karte befindet sich in einem Movieclip, da sie dupliziert werden soll und da sie zwei Seiten besitzt. Im ersten Frame befindet sich die Vorderseite der Karte – in diesem Beispiel ein blaues Rechteck. Im zweiten Frame befindet sich die Rückseite der Karte. Da die Karte durch Klicken auf die (linke) Maustaste umgedreht werden soll, muss das erste Rechteck auf der Vorderseite als Button ausgelegt sein. Zwar könnte man dies auch mit einem `onClipEvent(mouseDown)` lösen, aber die Lösung mit dem Button gestaltet sich einfacher.

Wenn der User nun auf eine Karte klickt, muss zuerst überprüft werden, ob sie die erste oder die zweite umgedrehte Karte ist. Falls es die erste Karte ist, übergibt diese ihren Wert in die Variable `_root.kartenwert`. Die zweite Karte setzt diesen Wert wieder auf 0.

Dadurch lässt sich dann auch feststellen, ob es sich um die erste oder die zweite Karte handelt, da die Variable `_root.kartenwert` bei der zweiten Karte einen Wert größer als o enthält.

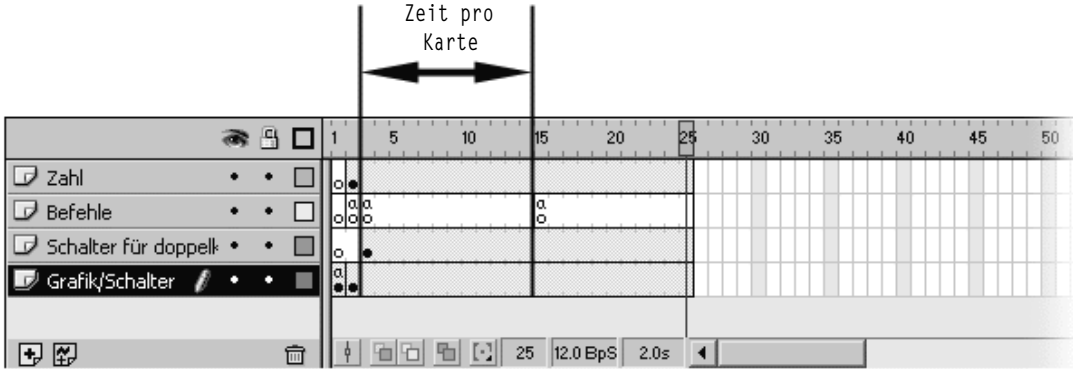
Falls es also die erste Karte ist, speichert diese ihren Wert in der Variable `_root.kartenwert`. Es muss aber auch noch zu erkennen sein, welche Karte umgedreht wurde. Dies lässt sich an den Werten nicht erkennen, da diese zufällig verteilt werden. Deshalb speichert die Karte ihren Instanznamen `_name` mit dem String "`_root.`" zusammen in der Variable `_root.karteeins`. Dadurch kann man anhand dieser Variablen direkt mit [ ] bzw. `eval()` auf die Eigenschaften der ersten Karte zugreifen, da diese genau den Pfad und den Instanznamen enthält. Danach springt der Movieclip zu dem zweiten Frame, in dem die Rückseite der Karte abgebildet wird. Die Rückseite ist nicht als Schalter ausgelegt, da sie keine Funktion besitzen muss.

Das ActionScript des Buttons einer Memorykarte:

```
on (press) {
    if (!_root.verriegelt) {
        if (_root.kartenwert>0) {
            _root.zug++;
            gotoAndPlay (3);
        } else {
            _root.karteeins = "_root."+_name;
            _root.kartenwert = kartennr;
            gotoAndStop (2);
        }
    }
}
```

Dies erste If-Abfrage mit der Bedingung `!_root.verriegelt` überprüft nur eine Variable, die verhindern soll, dass eine dritte Karte aufgedeckt wird, wenn bereits zwei Karten aufgedeckt sind. Die Variable `_root.verriegelt` ist also `true`, sobald zwei Karten aufgedeckt sind.

Wenn die erste Karte ausgewählt worden ist und eine zweite Karte angeklickt wird, wird wieder dieses ActionScript ausgeführt. Allerdings ist dann die Variable `_root.kartenwert` größer als 0, so dass `_root.zug` um 1 erhöht wird. Dies ist die Variable, welche die Anzahl der benötigten Spielzüge zählt. Anschließend springt der Movieclip zum dritten Frame und wird abgespielt. Im dritten Frame wird wieder die Rückseite der Memorykarte abgebildet, allerdings besitzt dieses Mal die Karte einen Button. Dadurch hat der User die Möglichkeit, durch einen Klick auf die Rückseite der zweiten Karte, die Karten wieder umzudrehen, ohne darauf zu warten, dass sie automatisch zurückgedreht werden. (Es nimmt immer eine gewisse Zeit in Anspruch, bis die dazwischen liegenden Frames in der Frameleiste abgespielt sind und der abschließende ActionScriptbefehl ausgeführt wird).



Diese Vorgehensweise ist die einfachste. Man hätte aber auch eine Variable auf den Wert von `getTimer()` setzen können, um die Differenz von `getTimer()` und der vorher gesetzten Variable so lange zu vergleichen, bis eine bestimmte Zeit abgelaufen ist.

Der Button, um die Zeit zu verkürzen, ist hier sehr simpel aufgebaut:

Zeitabfragen,
→ *6.1 Memory – Spieldauer-anzeige(getTimer).*

```
on (release) {
    gotoAndStop (15);
}
```

Bei Druck des Buttons springt das Movie dann einfach bis zu der Stelle mit dem ActionScript. Das ActionScript in dem Frame sieht wie folgt aus:

```
stop ();
_root.verrigelt = false;
_root.kartezwei = "_root."+_name;
_root.kartenwert -= kartennr;
if (_root.kartenwert == 0) {
      _root.befehl.spielerpunkte();
      removeMovieClip (_root.karteeins);
      removeMovieClip (_root.kartezwei);
      _root.anzahlderkarten -= 2;
      if (_root.anzahlderkarten == 0) {
              _root.timer.stopp = true;
              _root.gewonnen.gotoAndStop(2);
      }
} else {
      _root.befehl.spieler++;
      _root.befehl.spielersichtbar();
      gotoAndStop (1);
      _root.kartenwert = 0;
      with (eval(_root.karteeins)) {
              gotoAndStop(1);
      }
}
```

Hier wird geprüft, ob die Karten ungleich sind und wieder zurück gedreht werden müssen. Wenn die Karten gleich sind, müssen sie vom Spielfeld entfernt werden. In diesem Fall wird auch noch überprüft, ob es sich um die letzten zwei Karten handelt.

Wenn dieses ActionScript gestartet wird, wird als Erstes die Verriegelung der anderen Karten wieder entfernt mit `_root.verrigelt = false;`. Danach wird der Pfad zu der zweiten Karte in der Variablen `_root.kartezwei` gespeichert. Der Wert der zweiten Karte `kartennr` wird vom `_root.kartenwert` abgezogen, da der Wert der ersten Karte der Variablen `_root.kartenwert` entspricht. Wenn die Karten die gleichen Werte haben, ist `_root.kartenwert` gleich 0.

Deshalb wird in der folgenden If-Abfrage auch überprüft, ob dieser Wert 0 ist. Falls dies nicht zutrifft, müssen die Karten wieder zurückgedreht werden. Man muss dann diese Karten mit `gotoAndStop(1);` zu Frame 1 springen lassen, um wieder die Kartenvorderseite auf dem Bildschirm abzubilden. Die Befehle dafür befinden sich in dem else-Zweig. Zuerst kommen jedoch zwei Zeilen, die für die Spieleranzeige gebraucht werden; auf diese werden wir später noch einmal gesondert eingehen.

In der dritten Zeile folgt eine Goto-Anweisung für die zweite Karte, die dafür sorgt, dass die zweite Karte zurückgedreht wird. Darunter wird `_root.kartenwert` wieder auf 0 gesetzt, da dieser ungleich 0 war. Der Wert wurde benutzt, um zu überprüfen, ob schon eine Karte umgedreht worden ist und um die Werte der ersten und zweiten Karte zu vergleichen. Danach folgt ein `with()`, welches auf den Inhalt der Variablen `_root.karteeins` mit Hilfe von `eval()` zugreift. In der Variable `_root.karteeins` war der Pfad zu der ersten Karte gespeichert. Mit `with()` kann man von diesem Movieclip aus in einen anderen Movieclip Befehle übermitteln, z. B. wie hier `gotoAndStop(1);`. `eval()` wird gebraucht, da Flash auf den Inhalt der Variablen zugreifen muss. Ohne `eval()` würde der `with()`-Befehl versuchen, auf einen Movieclip mit dem Namen `_root.karteeins` zuzugreifen. Dieser Goto-Befehl sorgt auch wiederum nur dafür, dass die erste Karte zurückgedreht wird bzw. zu dem ersten Frame springt.

Falls die Karten gleich sein sollten, wird der erste Zweig der If-Abfrage ausgeführt:

```
_root.befehl.spielerpunkte();
removeMovieClip (_root.karteeins);
removeMovieClip (_root.kartezwei);
_root.anzahlderkarten -= 2;
if (_root.anzahlderkarten == 0) {
      _root.timer.stopp = true;
      _root.gewonnen.gotoAndStop(2);
}
```

Zuerst wird dort eine Funktion für die Spieleranzeige aufgerufen; diese erhöht die Punktzahl des aktuellen Spielers um 1. Anschließend werden die

zwei aufgedeckten Karten vom Spielfeld entfernt. Die entsprechenden Pfadeingaben zu den Movieclips stehen in den Variablen `_root.karteeins` und `_root.kartezwei`. Da sich nun zwei Karten weniger auf dem Spielfeld befinden, muss dies auch in einer Variablen vermerkt werden. Dies geschieht durch `_root.anzahlderkarten -= 2;`. Danach wird geprüft, ob sich noch Memorykarten auf dem Spielfeld befinden `if (_root.anzahlderkarten == 0)`. Falls das nicht der Fall ist, wird der Timer, der die Gesamtspielzeit zählt, gestoppt `root.timer.stopp = true;`. Dem Movieclip *_root.gewonnen* wird befohlen, zum zweiten Frame zu springen, wodurch der Schriftzug „Sie haben gewonnen“ angezeigt wird. `_root.gewonnen.gotoAndStop(2);`.

Die Farbe der Karten

Damit man die verschiedenen Memorykarten nicht nur an ihren unterschiedlichen Zahlen erkennen kann, sondern auch an unterschiedlichen Farben, haben wir ein ActionScript erstellt, das entsprechend des Kartenwertes die Farbe der Rückseite ändert. Das ActionScript befindet sich in der Frameleiste des Movies *karte* und wird aufgerufen, sobald eine der Rückseiten angezeigt wird.

```
farbe = new color(ukarte);
farbwert=kartennr add kartennr*4 add kartennr*kartennr add kartennr/100-100;
farbe.setRGB(farbwert);
```

Zuerst wird ein Colorobjekt erstellt. Als Argument muss man den Instanznamen des Movies angeben. *ukarte* ist der Instanzname des Movies der Kartenrückseite. Die Rückseite muss als Movie erstellt werden, da man sonst nicht das Colorobjekt auf diese anwenden kann. Das Colorobjekt hat den Namen *farbe*. Die Farbwerte kann man nun mit `.setRGB(Farbwerte) ändern`. Da die Karten in Abhängigkeit vom Kartenwert eine Farbe besitzen sollen, wird zuerst ein geeigneterer Farbwert über den Wert der Karte ermittelt. Die verschiedenen Ergebnisse werden durch `add` für den Farbwert als String aneinander gehängt. Der ermittelte Farbwert wird mit `setRGB` auf das Colorobjekt angewendet; dadurch ändert sich die Farbe des Movieclips *ukarte* entsprechend dem Wert der Karte.

Falls die zweite Karte aufgedeckt wird, wird an dieser Stelle noch die Variable `_root.verriegelt` gleich true gesetzt, um zu verhindern, dass während der Zeit, in der zwei Karten aufgedeckt sind, der User noch eine dritte Karte aufdeckt.

Spieldaueranzeige (getTimer())

Zur Messung der Gesamtspielzeit benutzt man die Funktion `getTimer()`. Diese ermittelt die Zeit in Millisekunden seit dem Start des SWFs.

Für den Timer benötigt man ein dynamisches Textfeld, um die ermittelte Zeit auszugeben. Dieses Textfeld stellt man am besten in ein Movie, da man für den Timer ein `onClipEvent()` braucht.

Beim Start des SWFs sollte man die Timerwerte alle auf o setzen:

```
onClipEvent (load) {
      sec = 0;
      min = 0;
      std = 0;
      startwert = 0;
      stopp = true;
}
```

`startwert` wird benutzt, um den Timer wieder auf o zu setzen. `startwert` wird beim Starten des Timers der aktuelle Wert von `getTimer()` übergeben, dadurch kann man diesen Wert von `GetTimer()` subtrahieren und erhält einen Wert, der bei o anfängt zu zählen.

`Stopp` wird benutzt, um den Timer zu stoppen, so dass dieser nur läuft, wenn ein Spiel im Gange ist. Das ActionScript des Timers sieht wie folgt aus:

```
 onClipEvent (enterFrame) {
      if (!stopp ) {
               sec = int(getTimer()/1000-startwert);
               if (sec>59) {
                        startwert += 60;
                        min++;
                        if (min>60) {
                                 min = 0;
                                 std++;
                                 if (std>24) {
                                          std = 0;
                                 }
                        }
               }
               if (  length(sec) < 2) {
                        sec = "0"+sec;
               }
               if (length(min) < 2) {
                        min = "0"+min;
               }
               if (length(std) < 2) {
                        std = "0"+std;
               }
               timer = std+" : "+min+" : "+sec;
      }
}
```

Hier wird zuerst überprüft, ob der Timer starten soll. Falls der Timer starten soll, ist die Variable `stopp=false` und die If-Abfrage erhält das Ergebnis `true`.

Als nächstes wird die Anzahl der Sekunden ermittelt.

```
sec = int(getTimer()/1000-startwert);
```

Da `getTimer()` den Wert in Millisekunden ausgibt, muss man diesen durch 1000 teilen und runden. Anschließend erfolgt die Subtraktion von `startwert`. `startwert` wird gleichzeitig mit dem start des Timers auf `getTimer()` gesetzt. Dies geschieht beim Drücken des Startbuttons.

Beim Drücken des Startbuttons werden die Werte des Timers auf 0 gesetzt und die Variable `_root.timer.stopp=false` gesetzt, damit der Timer zu laufen beginnt:

```
_root.timer.stopp=false;
_root.timer.sec=0;
_root.timer.min=0;
_root.timer.std=0;
...
..
.
_root.timer.startwert=int(getTimer()/1000);
```

In der ersten If-Abfrage beim Timer wird abgefragt, ob `sec` größer als 59 ist. Falls dies der Fall ist, wird `min` um 1 und `startwert` um 60 erhöht, wodurch `sec` um 60 kleiner wird.

```
if (sec>59) {
                        startwert += 60;
                        min++;
                        if (min>59) {
                                min = 0;
                                std++;
                                if (std>23) {
                                        std = 0;
                                }
                        }
                }
```

Der gleiche Vorgang wird bei `min` noch einmal wiederholt, nur dass dieses Mal nicht `startwert`, sondern `min` direkt um 60 gesenkt bzw. direkt auf 0 gesetzt wird. Zuletzt wird noch überprüft, ob `std` größer als 23 ist; falls dies der Fall ist, wird `std` zurück auf 0 gesetzt.

```
                if (  length(sec) < 2) {
                        sec = "0"+sec;
                }
                if (length(min) < 2) {
```

```
                min = "0"+min;
        }
        if (length(std) < 2) {
                std = "0"+std;
        }
        timer = std+" : "+min+" : "+sec;
}
```

Damit nun immer eine zweistellige Zahl ausgegeben wird, muss man mit `length()` die Länge der jeweiligen Zahl ermitteln. Wenn die Zahl nur einstellig ist, muss man dieser eine „0“ voranstellen:

```
sec = "0"+sec;
```

Zum Schluss wird die Ausgabe noch für das Textfeld aufbereitet und die Variablen mit Leerzeichen und Doppelpunkten versehen. Da diese Befehle in einem `onClipEvent(enterFrame)` stehen, wird diese Anweisung immer wiederholt und so dafür gesorgt, dass immer die aktuelle Spieldauer angezeigt wird.

Spieleranzeige

Zum Anzeigen der Spieler wird mit `.attachMovie` der Movieclip *Spieler* dupliziert bzw. an den Movieclip *Spielfeld* angehängt. Beim Drücken der Pfeiltaste wird eine Funktion aufgerufen, die einen Movieclip startet, der die Spieleranzahl erhöht. Beim Loslassen der Taste (`on(release)`) wird eine andere Funktion aufgerufen, die den Movieclip wieder stoppt und somit das Hochzählen beendet. Danach wird in einer for-Schleife das Movie *spieler* dupliziert und dem Movie s*pielfelder* angehängt.

```
spielerfelder.attachMovie("Spieler","spieler"+i, i);
```

Verknüpfungsnamen
→ *74 Soundeffekte.*

Der Befehl `attachMovie` arbeitet fast genauso wie der Befehl `duplicateMovieClip`. Auch bei `atachMovie()` wird von einem bestehenden Movie eine Kopie erzeugt. Allerdings muss sich das Movie in der Bibliothek befinden und einen Verknüpfungsnamen besitzen.

Vor dem `attachMovie`-Befehl steht der Pfad zu dem Movieclip, an dem das Movie, dessen Verknüpfungsname angegeben ist, angehängt werden soll.

```
.attachMovie("Spieler","spieler"+i, i);
```

Als Argumente werden der Verknüpfungsname, der neue Name für das Movie und die Tiefe für das neue Movie benötigt. Dies entspricht fast genau den Argumenten von `duplicateMovieclip`, bis auf die Tatsache, dass man nun einen Verknüpfungsnamen und keinen Instanznamen eingeben muss.

Das ActionScript des Buttons zum Erhöhen der Spielzahl lautet:

```
on (press) {
     if (spieleranzahl<200) {
              spieleranzahlalt = spieleranzahl;
              _root.befehl.count("_root.spieleranzahl.spieler-
              anzahl", "", 200);
     }
}
on (release, releaseOutside) {
     _root.befehl.stopcount();
     for (i=spieleranzahlalt+1; i<spieleranzahl+1; i++) {
              spielerfelder.attachMovie("Spieler", "spie-
ler"+i, i);
              spielerfelder["spieler"+i]._y = spielerfel-
der["spieler"+(i-1)]._y+15;
              spielerfelder["spieler"+i].Spieler = "Spieler"+i;
              spielerfelder["spieler"+i].Punkte = 0;
     }
     _root.spieleranzahl.spielerfelder._y = -445;
     _root.befehl.hide();
     _root.befehl.spielersichtbar();
     _root.befehl.pfeile();
}
```

Damit die for-Schleife die richtige Anzahl an Movieclips an den Movieclip *Spielerfelder* anhängt, muss man die Schleife unter Umständen mit einem höheren Anfangswert starten lassen. Falls schon fünf Spieler erzeugt wurden, und noch einmal vier erzeugt werden sollen, muss die Schleife viermal durchlaufen werden, obwohl insgesamt 9 Spieler hinterher zur Verfügung stehen müssen. Deshalb muss noch die alte Spielerzahl vor der Schleife gespeichert werden:

```
spieleranzahlalt = spieleranzahl;
```

Damit die Anzahl der Spieler nicht zu hoch wird, sollte man eine maximale Größe – hier 200 – festlegen.

Um den aktiven Spieler anzuzeigen, besitzt jeder Movieclip *Spieler* zwei Frames, in denen die Spieler und die Punkte einmal in blau und einmal in grau angezeigt werden. Die Funktion *spielersichtbar* lässt alle Movieclips ins zweite Frame springen und setzt anschließend den aktiven Movieclip ins erste Frame. Die Variable `_root.befehle.spieler` enthält immer die Zahl des aktiven Spielers.

```
function spielersichtbar () {
    spieleranzahl = _root.spieleranzahl.spieleranzahl;
if (spieler>spieleranzahl) {
    spieler = 1;
}
```

```
for (i=0; i<spieleranzahl+1; i++) {
    _root.spieleranzahl.spielerfelder["spieler"+i].goto-
    AndStop(2);
}
    _root.spieleranzahl.spielerfelder["spieler"+spieler].goto-
    AndStop(1);
}
```

Für die Punktezählung wird die Variable `punkte` bei dem jeweils aktiven Movieclip *Spieler* um eins erhöht:

```
function spielerpunkte () {
    _root.spieleranzahl.spielerfelder["spieler"+spie-
    ler].punkte += 1;
}
```

Um immer nur eine bestimmte Gruppe von Spielern anzuzeigen, falls mehr als darstellbar angezeigt werden sollen (Anzahl der Spieler > 17), haben wir mit einem ActionScript jeweils die Movies der Spieler, die außerhalb eines gewissen Bereiches liegen, unsichtbar gesetzt.

```
_root.spieleranzahl.spielerfelder["spieler"+i]._visible =
false;
```

Masken,
→72
Maskeneffekte.

Dies wäre unter Umständen einfacher und eleganter mit einer Maske zu lösen. Wenn Sie trotzdem einmal einen Blick in das ActionScript werfen möchten:

```
function hide () {
    hoehe = _root.spieleranzahl.spielerfelder._y;
    spielerzahl = _root.spieleranzahl.spieleranzahl;
    for (i=1; i-1<spielerzahl; i++) {
        if (hoehe+((i-1)*15)<-445) {
            _root.spieleranzahl.spielerfelder["spie-
            ler"+i]._visible = false;
        } else if (hoehe+((i-1)*15)>-200) {
            _root.spieleranzahl.spielerfelder["spie-
            ler"+i]._visible = false;
        } else {
            _root.spieleranzahl.spielerfelder["spie-
            ler"+i]._visible = true;
        }
    }
}
```

In der for-Schleife wird jeder Movieclip von den Spielern einzeln angesprochen; und nur wenn dieser innerhalb eines bestimmten Bereiches liegt `hoehe+((i-1)*15)>-200)` und `hoehe+((i-1)*15)<-445` wird dieser angezeigt:

```
_root.spieleranzahl.spielerfelder["spieler"+i]._visible = true;
```

Die Berechnung für die Höhe des jeweiligen Movieclips *spieler* muss variabel sein, da der übergeordnete Movieclip *spielerfelder* auf der Y-Achse verschoben wird, um andere Spieler anzeigen zu können.

Anzeige der Uhrzeit

Die Anzeige der Uhrzeit ist bei Flash 5 sehr einfach zu realisieren: Man muss zuerst ein date-Objekt erzeugen.

```
zeit = new date();
```

Dieses hat hier den Namen *zeit,* und über die Funktionen `getseconds();` `getminutes();gethours();` kann man die Sekunden, Minuten und Stunden abfragen:

```
sec = zeit.getseconds();
min = zeit.getminutes();
std = zeit.gethours();
```

Anschließend erfolgt die gleiche Überprüfung wie bei der *Spieldaueranzeige*. Es wird geprüft, ob die Zahlen einstellig sind, und falls dies zutrifft, wird diesen eine „0“ vorangestellt.

```
if (length(sec)<2) {
    sec = "0"+sec;
}
if (length(min)<2) {
    min = "0"+min;
}
if (length(std)<2) {
    std = "0"+std;
}
```

Zum Schluss werden die einzelnen Variablen zu einem String zusammengefasst und dem Textfeld *Uhrzeit* zugeordnet:

```
Uhrzeit = std+" : "+min+" : "+sec;
```

6.2 Bewegung

Ordner Bewegung, Datei: 2bewegun.fla

Bewegungen – z. B. eines Balls – werden in Flash meist mit Tweenings realisiert. Das Problem dabei ist, dass man auf die Tweenings mit ActionScript keinen Einfluss hat. Deshalb ist dieser Lösungsweg nicht sehr hilfreich.

Mit ActionScript kann man nur Movies beeinflussen, deshalb erstellt man einen Movieclip mit einem Ball. Diesen Movieclip kann man per ActionScript bewegen. Da die Aktionen des ActionScripts immer wieder ausgeführt werden müssen, erstellt man dieses in einem `onClipEvent(enterFrame)`.

Zuerst wird eine Anfangsgeschwindigkeit festgelegt:

```
ybewegung = -1;
xbewegung = 1;
```

Dies geschieht in einem `onClipEvent(load)`, da dies eine einmalige Aktion ist. Nachdem die Anfangswerte gesetzt sind, muss man nun dafür sorgen, dass der Movieclip *Ball* sich entsprechend bewegt. Auf die Eigenschaften des Movieclips kann man mit ActionScript problemlos zu greifen, hierbei relevant sind erst einmal nur die X- und Y-Position des Movieclips. Dies wird durch das folgende ActionScript realisiert:

```
function beschleunigung () {
    this._y = this._y-ybewegung;
    this._x = this._x-xbewegung;
}
```

Hier wird der Movieclip *Ball* (*this*) immer um die entsprechende Pixelzahl versetzt. Dies geht so lange gut, bis der Ball sich aus dem Sichtfeld des Bildschirms heraus bewegt. Setzen Sie darum Grenzen und kehren Sie an diesen Grenzen die Bewegungsrichtung des Balles um.

Begrenzung der Bewegung

Zuerst muss man die Grenzwerte setzen. Da dies wieder einmalig geschieht, steht der Befehl in einem `onClipEvent(load)`:

```
linkerrand = 50;
rechterrand = 750-this._width;
obererrand = 50;
untererrand = 550-this._height;
```

Da das SWF später eine Ausgabegröße von 800 x 600 Pixel haben wird Strg+M, muss man von der rechten und von der unteren Seite 50 plus den Durchmesser des Balles subtrahieren. Zu der linken und zu der oberen Seite muss man 50 addieren, wenn man einen Rand von 50 erhalten möchte. Nachdem Sie die Grenzwerte gesetzt haben, erzeugen Sie noch eine geeignete If-Abfrage:

```
function inabox () {
    if (this._x<linkerrand) {
    } else if (this._x>rechterrand) {
    }
    if (this._y<obererrand) {
    } else if ( this._y>untererrand) {
    }
}
```

Diese If-Abfragen vergleichen, ob sich der Ball außerhalb der gesetzten Grenzen befindet. Da der Ball nicht gleichzeitig die linke und die rechte Grenze überschreiten kann, kann man dies in einer If-else-if-Abfrage verschachteln. Das Gleiche gilt für die obere und untere Grenze. Man darf aber keinesfalls alles der in If-else-if-Abfrage verschachteln, da es gut möglich ist, dass der Ball zwei Grenzen gleichzeitig überschreitet, wenn er sich genau in der Ecke befindet.

Nachdem Sie nun die If-Bedingung festgelegt haben, sorgen Sie noch dafür, dass der Ball seine Richtung ändert und nicht weiter seine Grenzen überschreitet. Die Richtungsänderung erfolgt über einen simplen Vorzeichenwechsel der Bewegungsvariablen, bei X also `xbewegung` und bei Y `ybewegung`:

```
xbewegung = -xbewegung;
ybewegung = -ybewegung;
```

Dies fügt man dann in die eben erstellte If-Abfrage ein und erhält dann folgendes ActionScript:

```
function inabox () {
      if (this._x<linkerrand) {
               xbewegung = -xbewegung;
      } else if (this._x>rechterrand) {
               xbewegung = -xbewegung;
      }
      if (this._y<obererrand) {
               ybewegung = -ybewegung;
      } else if ( this._y>untererrand) {
               ybewegung = -ybewegung;
      }
}
```

Korrektur der Grenze

Dieser Aufbau funktioniert zwar, hat aber den Nachteil, dass das ActionScript erst eingreift, wenn der Ball die Grenze schon überschritten hat. Und umso schneller der Ball ist, desto weiter überquert er auch die Grenze, was aber nur bei genauem Hinsehen auffällt. Die Grenze insgesamt weiter zur Mitte zu verlegen, funktioniert nur bei konstanter Fluggeschwindigkeit des Balls. Da seine Geschwindigkeit aber variiert, ändert man die Abfrage nicht, sondern errechnet den Wert, den der Ball erreichen würde, wenn er an der Grenze abgeprallt wäre.

1. Setzen Sie den Ball auf die neuen Bewegungswerte,
2. Prüfen Sie, ob der Ball außerhalb der Grenzwerte liegt.
   Falls dies zutrifft:
   - Kehren Sie die Bewegung des Balles um.
   - Setzen Sie den Ball auf die korrigierten Positionswerte.

Wenn der Ball beispielsweise die linke Grenze überschreitet, so errechnet man den Wert, um den der Ball diese Grenze überschritten hat, mit `linkerrand-this._x`. Wenn das Ergebnis wieder zur aktuellen Position des Balls addiert wird, liegt dieser genau an der Grenze. Doch der Wert, um den er die Grenze überschritten hat, verschwindet einfach. Da der Ball aber von der Grenze abprallt, und seine Bewegung in umgekehrter Richtung fortsetzt, verdoppelt man den eben errechneten Wert `(linkerrand-this._x)`, um dies real darzustellen. Die komplette Abfrage sieht dann so aus:

```
function inabox () {
      if (this._x<linkerrand) {
              xbewegung = -xbewegung;
              this._x += (linkerrand-this._x)*2;
      } else if (this._x>rechterrand) {
              xbewegung = -xbewegung;
              this._x -= (this._x-rechterrand)*2;
      }
      if (this._y<obererrand) {
              ybewegung = -ybewegung;
              this._y += (obererrand-this._y)*2;
      } else if ( this._y>untererrand) {
              ybewegung = -ybewegung;
              this._y -= (this._y-untererrand)*2;
      }
}
```

Einfluss des Users

▶ **Ball bewegen**

Der Ball bewegt sich, aber der User hat keinen Einfluss auf die Bewegungsrichtung. Um dies zu ändern, legt man die Grafik des Balls, die sich in dem Movie *ball* befindet, als Button aus, so dass der User die Möglichkeit erhält, den Ball anzuklicken. Damit der Ball auf einen Klick hin bewegt werden kann (Ziehen bzw. drag), muss man den Movieclip *ball* mit `startDrag()` an die Maus koppeln. Da der Ball hierbei auch nicht die Grenzen überschreiten darf, gibt man diese dabei mit an.

```
on (press) {
startDrag ("_root.ball", false, linkerrand, obererrand, rech-
terrand, untererrand);
dragball = true;
}
```

Beim Loslassen der Maustaste muss der Drag-Befehl wieder aufgehoben werden.

Begrenzung von startDrag,
→ *74 Soundeffekte.*

```
on (release) {
stopDrag ();
dragball = false;
}
```

Während des Drags darf der Ball nicht weiter verschoben werden. Deshalb werden die Funktionen für die Bewegung und für die Grenzberechnung in dem aktuellen Zeitraum nicht mehr ausgeführt.

```
if (!dragball) {
      beschleunigung();
      inabox();
}
```

▶ **Indirekte Steuerung der Geschwindigkeit**

Um dem User zu ermöglichen, die Geschwindigkeit direkt zu beeinflussen, erstellt man zwei Textfelder, in denen er die Geschwindigkeit eingeben kann und per Button übermittelt.

Bei einem Spiel wird dies aber eher selten der Fall sein. Deshalb haben wir hier die Bewegung in Abhängigkeit zur Positionsänderung der Maus berechnet.

Dafür speichert man zuerst die alte Position der Maus über die Variablen `altx` und `alty`:

```
on (press) {
      altx = this._x;
      alty = this._y;
      dragball = true;
      startDrag ("_root.ball", false, linkerrand, obererrand,
rechterrand, untererrand);
}
```

→ *70 Mauseffekte/ Maustrailer – Mousekoordinaten.*

Hier wird die Position des Balls abgefragt und in den Variablen `altx` und `alty` gespeichert. Wir haben hierfür die Position des Balls abgefragt, da diese bei einem Drag der Position des Mauszeigers entspricht. Man kann aber genauso gut die Eigenschaften `_xmouse` und `_ymouse` benutzen.

Wenn man nun bei `on(release)` die Koordinaten des Balls wieder abfrägt, kann man aus der Differenz der alten und der neuen Koordinaten seine Geschwindigkeit errechnen.

Die Berechnung der Geschwindigkeit basiert aber immer auf den Ursprungskoordinaten des Punktes, an denen der Ball an die Maus gehängt wurde bzw. an denen der User auf den Ball geklickt hat. Jeder erwartet allerdings ein anderes Verhalten des Balles: nämlich dass der Ball in die Richtung fliegt, in die man ihn bewegt, egal an welcher Position man den Ball aufgenommen hat. Damit dies funktioniert, müssen die Werte von `altx` und

alty immer angeglichen werden. Hierfür baut man die Abfrage am besten wieder in ein onClipEvent(enterFrame). Sobald der Ball angeklickt wird, muss diese Abfrage starten und die Position des Balls aktualisieren. Da die Abfrage, ob der Ball an den Mauszeiger angehängt wurde, schon existiert, kann man dies direkt mit in die Abfrage einbauen:

```
onClipEvent (enterFrame) {
      if (!dragball) {
            beschleunigung();
            inabox();
      } else {
            mausbewegung();
      }
}
```

Zuvor wurde die Funktion *Mausbewegung* mit Hilfe eines onClip-Event(load) erstellt. Die Funktion sieht wie folgt aus:

```
function mausbewegung () {
      altx = this._x;
      alty = this._y;
}
```

Dadurch wird die alte Ballposition immer in einer Variable gespeichert. Mit Hilfe dieser Variablen bildet man, wie eben schon erwähnt, die Differenz zu der neuen Position und leitet daraus die Geschwindigkeit des Balls ab.

Nun kann es aber passieren, dass der User den Ball zu sehr beschleunigt. Führen Sie wieder Grenzwerte ein, um ihn zu bremsen.

Diese werden wieder im Handler onClipEvent(load) eingefügt. Achten Sie hierbei auf eine eindeutige Variablenbezeichnung oder fügen Sie direkt Kommentare in das ActionScript ein, weil ihre Funktion eventuell nicht sofort ersichtlich wird.

```
onClipEvent (load) {
      maxbewegungmaus = 10;
      minbewegungmaus = -10;
}
```

Nach dem Festlegen der Grenzwerte kann man die Differenz bilden und daraus die Geschwindigkeit errechnen. Falls die Geschwindigkeit die Grenzwerte überschreitet, muss man sie anpassen. Die Überprüfung realisiert man mit einer If-Abfrage:

```
on (release) {
      xbewegung = altx-this._x;
      ybewegung = alty-this._y;
      if (xbewegung>maxbewegungmaus) {
            xbewegung = maxbewegungmaus;
      }
```

```
        if (xbewegung<minbewegungmaus) {
                xbewegung = minbewegungmaus;
        }
        if (ybewegung>maxbewegungmaus) {
                ybewegung = maxbewegungmaus;
        }
        if (ybewegung<minbewegungmaus) {
                ybewegung = minbewegungmaus;
        }
        dragball = false;
        stopDrag ();
}
```

6.3 Fallbeschleunigung/Gravitation

Grundüberlegungen

Ordner Bewegung, Datei: 3Grafit.fla

Mit ballistischen Berechnungen können Sie Ihrem Ball eine natürliche Bewegung verleihen.

Ein Ball, der gegen ein Hindernis prallt, wird mit verminderter Geschwindigkeit in die entgegengesetzte Richtung zurückgeschleudert.
Bei einem Aufprall entsteht immer ein Energieverlust, was in die Berechnung miteinfließen muss.

Zum anderen ergibt sich während des Fluges des Balls ein Energieverlust durch den Luftwiderstand, so dass die Werte für die *x*- und *ybewegung* kleiner werden.

Rollt der Ball am Boden entlang, so führt der Reibungswiderstand des „Bodens“ wiederum zu einem Geschwindigkeitsverlust.

Die ganzen Berechnungen werden hier nur annährend ausgeführt, so dass die Bewegung dem Benutzer zwar realistisch erscheint, aber doch von den Naturgesetzen abweicht.

Steuerung der Bewegung

Steuerung und Bewegung verändern sich im Vergleich zu dem vorangegangenen Beispiel 6.3 „Bewegung eines Balls“ nicht. Die Berechnung wird nur noch um die Einflüsse der Gravitation ergänzt.

Berechnung der Gravitation

Dieses Movie kann schlecht als Beispiel für ballistische Berechnungen dienen: Wir stellen lediglich die Bewegung eines Balls dar, die der Flugbahn eines Balls unter Gravitationseinflüssen ähnlich ist.

Da wir bei der Bewegung des Balls immer auf die Geschwindigkeit direkt zugegriffen haben, benutzt man für die Gravitation am besten die Formel:

$$V_g = g \cdot t$$

Fallgeschwindigkeit = Fallbeschleunigung · Zeit

Die Zeit wird wieder über `getTimer()` ermittelt und mit einer zusätzlichen Variable wieder auf den Startwert zurückgerechnet.

→ 6.1 Memory – Spieldaueranzeige.

Die Fallgeschwindigkeit wird in der Funktion `grav()` errechnet.

```
function grav () {
      tanfang = getTimer()-tzwischen;
      yfall = (g*(tanfang/1000));
}
```

Mit diesem ActionScript wird zuerst die Differenz von `getTimer()` und `twischen` ermittelt, damit man eine Variable erhält, die man auf den Wert von `getTimer()` setzt und so `tanfang` wieder von 0 beginnt, und der Timer wieder bei 0 zu zählen beginnt. `yfall` wird dann nach der oben beschriebenen Formel berechnet. Die Fallgeschwindigkeit muss dann zu der bisherigen `ybewegung` addiert werden.

```
function beschleunigung () {
      this._y = this._y-ybewegung+yfall;
      this._x = this._x-xbewegung;
}
```

Der Ball müsste nun von der Schwerkraft angezogen nach unten fallen, vom Boden abprallen, wieder nach oben fliegen und so weiter. Da der Ball keine Energieverluste erfährt, würde er sich unendlich weiter bewegen. Da die Berechnungen ungenau sind, wird der Ball sogar noch immer schneller. Dies ist jedoch unerheblich, da in der weiteren Berechnung die Verluste eingebunden werden, wodurch die Abweichung nicht mehr auffällt.

Abbremsung durch den Reibungswiderstand der umgebenden Masse

Im Normalfall geht man davon aus, dass ein Ball durch die Luft fliegt. Die Abbremsung bzw. Abdämpfung der Geschwindigkeit durch den Luftwiderstand wäre dementsprechend relativ gering anzusetzen. Wir haben hierfür den Wert 0,998 eingestellt. Es handelt sich dabei um einen Faktor, der lediglich in etwa die Wirkung simulieren soll, die der Luftwiderstand tatsächlich auf den Ball hat. Hier eine exakte Berechnung einzubinden, ist sehr komplex, da die Dämpfung von Größe und Form der Oberfläche und von der Geschwindigkeit abhängt. Bei Wasser wäre dieser Faktor sehr viel kleiner anzusetzen; 1 würde für den luftleeren Raum angenommen.

Die Einbindung in das ActionScript erfolgt durch folgende Funktion, die bei jeder Berechnung mitaufgerufen wird und die die Geschwindigkeit entsprechend dem Dämpfungsfaktor verringert.

```
function daempfungdesballs () {
      daempfung = _root.daempfung;
      ybewegung = ybewegung*daempfung;
      xbewegung = xbewegung*daempfung;
}
```

Da wir die `daempfung` als Textfeld angelegt haben, wird dieser Wert zuerst aus dem Textfeld gelesen und in den Movieclip übertragen.

```
daempfung = _root.daempfung;
```

Berührung des Bodens

```
} else if (this._y>untererrand) {
      ybewegung = -ybewegung+yfall;
      this._y -= (this._y-untererrand);
      ybewegung *= aufprall;
      xbewegung *= Reibungswiderstand ;
      if (ybewegung<1 && ybewegung>-1&&this._y>untererrand-5) {
              ybewegung = 0;
      }
              tzwischen = getTimer();
              yfall = 0;
  }
```

Da jeder Aufprall auf den Boden einen Geschwindigkeitsverlust mit sich bringt, muss man die Geschwindigkeit im Moment des Aufpralls verringern: `ybewegung *= aufprall;`. In unserem Beispiel wird die Geschwindigkeit um den Faktor 0,5 verkleinert. Die ersten beiden Zeilen berechnen die aktuelle Position des Balls:

```
ybewegung = -ybewegung+yfall;
this._y -= (this._y-untererrand);
```

Wenn `ybewegung = 0` ist, würde der Ball auf dem Boden entlang rollen. Dies würde auch bedeuten, dass die If-Abfrage `} else if (this._y>untererrand) {` immer wahr ist, deshalb steht dort auch `xbewegung *= Reibungswiderstand ;`, da ein rollender Ball einem Reibungswiderstand ausgesetzt ist.

Da die Berechnung der Gravitationswirkung ungenau ist, wird ein Offset-Abgleich eingebaut. Dies bedeutet, dass wenn ein bestimmter Wert unterschritten wird, dieser direkt auf 0 gesetzt wird.

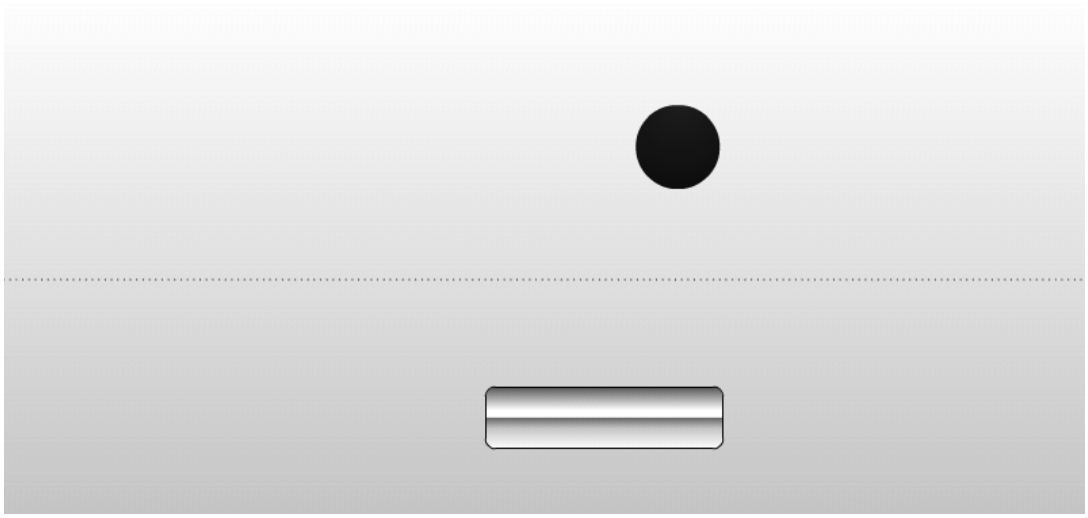
```
if (ybewegung<1 && ybewegung>-1&&this._y>untererrand-5) {
      ybewegung = 0;
}
```

Sobald `ybewegung` zwischen –1 und 1 liegt, und der Ball sich in Bodennähe befindet, wird `ybewegung` gleich 0 gesetzt. Wenn Sie dies unterlassen, bewegt sich der Ball am Boden immer ein wenig weiter, da die Berechnung für `ybewegung` nie direkt 0 ergibt.

Da beim Auftreffen des Balls am Boden die Fallbeschleunigung gestoppt wird bzw. die Fallgeschwindigkeit wieder neu berechnet werden muss, wird `tzwischen = getTimer();` und `yfall=0;` gesetzt. Genau das gleiche muss passieren, wenn der Ball von der Maus bewegt wird, da der Wert der alten Fallgeschwindigkeit gelöscht werden muss.

6.4 Einbinden von zusätzlichen Movies (Objekten)

maximal erreichte Höhe = 22
Haftung = 0.1
Abprallverhältnis = 0.5
g(m/s^2) = 9.81
Masse = 35
Dämpfung = 0.999
Aufprallverlust = 0.5



Steuerung des Movies durch Tastaturabfrage

Ordner Bewegung, Datei: 4zusatz.fla

Damit der User ein Movie (Objekt) mit der Tastatur bewegen kann, muss man unter Flash 5 zuerst die Tasten überprüfen. Dafür gibt es anders als bei Flash 4 nun ein Key-Objekt, dessen Werte bei gedrückter Taste *true* und bei losgelassener Taste *false* sind.

Die Abfrage der Tastatur sieht wie folgt aus:

```
if (Key.isDown(Key.DOWN)) {
}
```

onClpiEvent()
→ *Kapitel 2: Neuerungen in Flash 5.*

Die angesprochene Taste heißt *DOWN* [↓] . Wenn diese Taste gedrückt wird, werden die in der If-Bedienung stehenden Befehle ausgeführt. Da diese Abfrage regelmäßig erfolgen muss, wird dies in einem `OnClipEvent (enterFrame)` erstellt. Es gibt auch ein `onClipEvent (keyDown)`, das ausgeführt wird, sobald eine Taste gedrückt wird. Hierbei ergibt sich aber (ähnlich wie bei der Bewegung des Cursors auf dem Bildschirm) nach der ersten Aktion eine Verzögerung, die bei dieser Art der Steuerung natürlich unerwünscht ist.

Mit dieser Abfrage kann der User nun die Walze steuern, wofür man aber noch die restlichen Pfeiltasten abfragen muss.

```
onClipEvent (enterFrame) {
      if (_root.walze._y>0+linkerrand && Key.isDown(Key.Down)) {
            ygeschwindigkeit++;
      }
      if (_root.walze._y+_root.walze._width<rechterrand &&
      Key.isDown(Key.UP)) {
            ygeschwindigkeit--;
      }
      if (_root.walze._x>0+linkerrand && Key.isDown(Key.LEFT)) {
            xgeschwindigkeit++;
      }
      if (_root.walze._x+_root.walze._width<rechterrand &&
      Key.isDown(Key.RIGHT)) {
            xgeschwindigkeit--;
      }
}
```

Abhängig davon, welche Taste gedrückt wird, wird die X- bzw. Y-Geschwindigkeit der Walze erhöht bzw. gesenkt. Die If-Abfragen beinhalten noch die maximalen Grenzen für die Walze. Sie ähneln den If-Abfragen für den Ball. Die Tastaturabfrage erfolgt über die Funktion `Key.isDown()` und die entsprechenden Taste – wie hier z. B. `Key.isDown(Key.LEFT)` für die linke Pfeiltaste.

Damit sich der Movieclip *walze* mit der zugeordneten Geschwindigkeit in die richtige Richtung bewegt, muss man diesen, wie den Ball auch, auf die jeweilige _x- und _y-Position setzen.

```
_root.walze._x -= xgeschwindigkeit;
xgeschwindigkeit *= 0.95;
_root.walze._y += ygeschwindigkeit;
ygeschwindigkeit *= 0.95;
```

Der Multiplikator dient zur Abbremsung der Walze, wie zuvor bei dem Ball die Dämpfung.

Abgrenzung der Bewegung

Damit die Walze während der Bewegung nicht vom Bildschirm verschwindet, müssen die Randpositionen wieder einzeln abgefragt werden, da die obige Randbegrenzung nur die Tastatureingabe stoppt.

```
        if (_root.walze._x<linkerrand) {
                xgeschwindigkeit = -xgeschwindigkeit;
                _root.walze._x = 50.1;
        } else if (_root.walze._x+_root.walze._width>rechterrand)
{
                xgeschwindigkeit = -xgeschwindigkeit;
                _root.walze._x = rechterrand-0.1-
_root.walze._width;
        }
        if (_root.walze._y<400) {
                ygeschwindigkeit = -ygeschwindigkeit;
                _root.walze._y = 400;
        } else if (_y>untererrand-50) {
                ygeschwindigkeit = -ygeschwindigkeit;
                _root.walze._y = untererrand-50;
        }
```

Das Grundgerüst der Abfrage entspricht genau derjenigen des Balls. Mit der Ausnahme, dass sich die Randwerte verändert haben. Die Werte mussten um 0,1 korrigiert werden, da die Walze sonst nicht an der Wand still anliegen würde.

Kollisionsdetection

Da bei Flash 5 die neue `hittest()`-Funktion groß angepriesen wurde, haben wir diese hier benutzt, um zu testen, wann der Ball die Walze berührt. Der erste auffällige Nachteil ist der, dass diese Funktion erst auftritt, nachdem der Ball die Walze berührt hat. Deshalb ist es für komplexe Abfragen dieser Art sinnvoller, doch die einzelnen Punkte der Walze abzufragen: also einfach die _y-Position und den Bereich zwischen _x und _x+ `_width`. Bei dem Ball-Beispiel wäre dies komplexer. Da der Ball rund ist, müsste man für ihn den Mittelpunkt errechnen, über den Radius mit Hilfe von Sinus und Kosinus nacheinander in einer for-Schleife die einzelnen Punkte errechnen und mit der Walze vergleichen. Zudem müsste man noch für den nicht frontalen Aufprall über den Hebelarm den Abprallwinkel errechnen.

Diese Abfrage finden Sie demnächst auf unserer Internetseite www.DieFlasher.de,
→ Abschnitt 10: Anhang.

Darauf haben wir hier bewusst verzichtet, um die neue `hittest()`-Funktion zu demonstrieren. An Stelle des Hittests können Sie aber genauso gut ein eigens hierfür erstelltes ActionScript einbinden. In den vorherigen Kapiteln wurde die Programmierung physikalischer Formeln in Flash ja des öfteren erläutert.

Setzen Sie die Funktion `hittest()` in Verbindung mit einer If-Abfrage ein:

```
if (_root.walze, hittest(_root.ball)) {
```

Diese wird immer wahr, wenn ein Movieclip den anderen überlappt. Da Flash aber Movies immer rechteckig markiert, sieht der Movieclip des Balls nicht mehr rund aus.

Diese Funktion wird also auch wahr, wenn eine Kante des Balls die Walze überlappt. In diesem Fall addieren Sie die Kräfte und verteilen sie nach den gegebenen Bedingungen wieder.

Energieübertragung

Für die Berechnung der Aufprallenergie des Balls gilt:

$$M \cdot V$$

Masse · Geschwindigkeit

Das Gleiche gilt für die Energie der Walze. So kann man die Energie des Balls und die der Walze addieren und damit die Gesamtenergie ausrechnen. Die Gesamtenergie teilt man anschließend wieder auf: In die Bestimmung des Abprallverhaltens des Balls geht einerseits ein, wie sich das Gewicht von Ball und Walze zueinander verhalten, andererseits die vorgesehene Abprallstärke. Geht der Wert gegen 1, so entspricht dies einer zunehmenden Verlagerung der Energie auf den Ball – bis dahin, dass die Walze sich anschließend gar nicht mehr bewegt.

```
altenergieball = _root.masse*-(_root.ball.ybewegung +
_root.ball.yfall);
altenergiewalze = walzenmasse*ygeschwindigkeit;
gesamtenergie = -(altenergiewalze+altenergieball);
energieball = gesamtenergie*(_root.abprall);
energiewalze = gesamtenergie*(1-_root.abprall);
_root.ball.ybewegung = energieball/_root.masse;
ygeschwindigkeit = energiewalze/walzenmasse;
```

Danach multipliziert man die Geschwindigkeiten noch mit dem Aufprall-bedingten Verlust und setzt den Startwert für die Gravitationsberechnung wieder auf 0.

Die Zeile mit dem _x-Werten ist die Haftung bzw. der Spinn des Balls, der durch den Aufprall auf die Walze erzeugt wird. Dies dient dazu, den Ball durch „Anscheiden“ mit der Walze in der _x-Bewegung zu beeinflussen.

```
ygeschwindigkeit *= _root.aufprall;
      with (_root.ball) {
      xbewegung += -(_root.haftung*_root.walze.xgeschwindig-
      keit);
      ybewegung *= _root.aufprall;
      // ball oben auf die walze setzen
      _y = _root.walze._y-_height-1;
      yfall = 0;
      tzwischen = getTimer();
}
```

Mit dem `with`-Befehl kann man direkt auf einen anderen Movieclip zugreifen. Dies ist zwar auch mit der Punkt-Syntax möglich; aber bei mehreren Befehlen trägt dies zu einer kleinen Performancesteigerung bei. Dies spielt allerdings erst eine Rolle, wenn man diese Befehle sehr oft während eines Frames in einer Schleife aufruft.

Als Argument wird der `with`-Befehl mit Instanznamen des anzusprechenden Movieclip aufgerufen, hier: *_root.ball*.

Die benötigten Variablen/Konstanten werden zu Beginn einmal in einem `onClipEvent(load)` gesetzt. Die restlichen Aktionen werden alle in einem `onClipEvent(enterFrame)` aufgerufen.

Im Folgenden das komplette ActionScript der Walze auf einen Blick:

```
onClipEvent (load) {
      linkerrand = 50;
      rechterrand = 750;
      untererrand = 650-this._height;
      xgeschwindigkeit = 0;
      ygeschwindigkeit = 0;
      walzenmasse = 50;
}
onClipEvent (enterFrame) {
      if (_root.walze, hittest(_root.ball)) {
            altenergieball = _root.masse*-(_root.ball.ybewe-
            gung+_root.ball.yfall);
            altenergiewalze = walzenmasse*ygeschwindigkeit;
            gesamtenergie = -(altenergiewalze+altenergie-
            ball);
            energieball = gesamtenergie*(_root.abprall);
            energiewalze = gesamtenergie*(1-_root.abprall);
            _root.ball.ybewegung = energieball/_root.masse;
            ygeschwindigkeit = energiewalze/walzenmasse;
            ygeschwindigkeit *= _root.aufprall;
            with (_root.ball) {
                  xbewegung += -(_root.haf-
                  tung*_root.walze.xgeschwindigkeit);
```

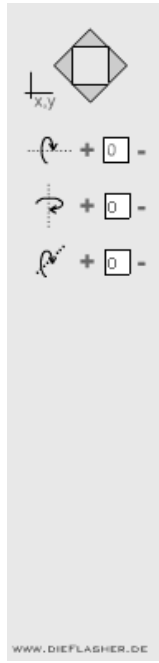
```
                    ybewegung *= _root.aufprall;
                    // ball oben auf die walze setzen
                    _y = _root.walze._y-_height-1;
                    yfall = 0;
                    tzwischen = getTimer();
                }
        }
        _root.walze._x -= xgeschwindigkeit;
        xgeschwindigkeit *= 0.95;
        _root.walze._y += ygeschwindigkeit;
        ygeschwindigkeit *= 0.95;
        if (_root.walze._x<linkerrand) {
                xgeschwindigkeit = -xgeschwindigkeit;
                _root.walze._x = 50.1;
        } else if (_root.walze._x+_root.walze._width>rechterrand)
{
                xgeschwindigkeit = -xgeschwindigkeit;
                _root.walze._x = rechterrand-0.1-
                _root.walze._width;
        }
        if (_root.walze._y<400) {
                ygeschwindigkeit = -ygeschwindigkeit;
                _root.walze._y = 400;
        } else if (_y>untererrand-50) {
                ygeschwindigkeit = -ygeschwindigkeit;
                _root.walze._y = untererrand-50;
        }
        if (_root.walze._y>0+linkerrand && Key.isDown(Key.Down)) {
                ygeschwindigkeit++;
        }
        if (_root.walze._y+_root.walze._width<rechterrand &&
        Key.isDown(Key.UP)) {
                ygeschwindigkeit--;
        }
        if (_root.walze._x>0+linkerrand && Key.isDown(Key.LEFT)) {
                xgeschwindigkeit++;
        }
        if (_root.walze._x+_root.walze._width<rechterrand &&
        Key.isDown(Key.RIGHT)) {
                xgeschwindigkeit--;
        }
}
```

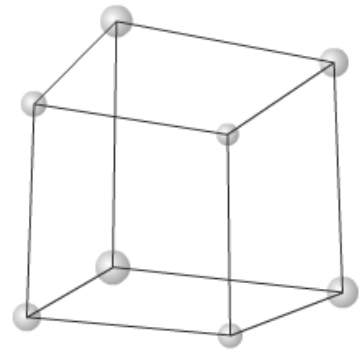
Die Werte auf der Hauptbühne müssen immer wieder neu abgefragt werden, da diese aktuell vom Benutzer geändert werden können, was aber nor-

malerweise bei einem Spiel nicht der Fall ist. Wir haben dies so gestaltet, damit Sie diese Werte verändern und direkt den Unterschied erkennen können. Dabei sollten Sie aber darauf achten, dass die Werte während ihrer Änderung niemals auf 0 sinken. Genauso wenig sollten Werten, die kleiner als 1 sind, Werte über 1 zugewiesen werden, da es sich um Faktoren handelt, die dazu dienen, eine Bewegung zu bestimmten Zeitpunkten abzubremsen.

6.5 Rotierender 3D-Würfel

Ordner 3DWuerfe, Datei: 3D.fla bzw. 3DFlash5.fla





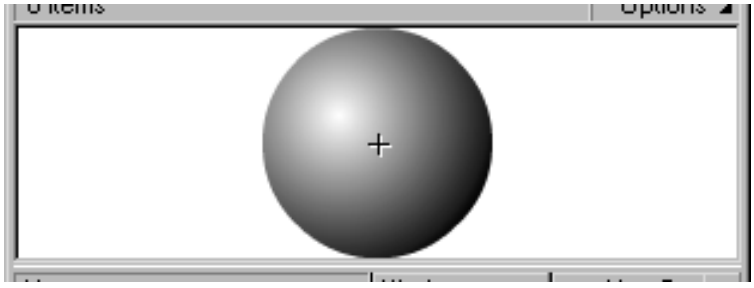
Möchten Sie in Flash – gegebenenfalls unter dem Aspekt der Spieleprogrammierung – immer wieder neue Effekte auf der Seite zeigen, bietet sich der folgende Effekt an. Da das Internet nicht nur informiert, sondern den Besucher auch unterhält, ist es naheliegend, interaktive Spielereien auf Ihrer Site unterzubringen. Sie bieten nicht nur Einblicke in den technischen Standard Ihrer Site, sie bieten sogar Einflussmöglichkeiten und Interaktivität. Die Informationspflicht ist aber dessen ungeachtet oberstes Gebot im Internet und darf nicht von Effekten zugedeckt werden.

Da Sie mit Flash mittlerweile etwas vertrauter sein dürften, werden wir alle benötigten Elemente auflisten. Sollten Sie als Quereinsteiger Verständnisprobleme haben, können Sie in den entsprechenden vorangegangenen Kapiteln nachschlagen.

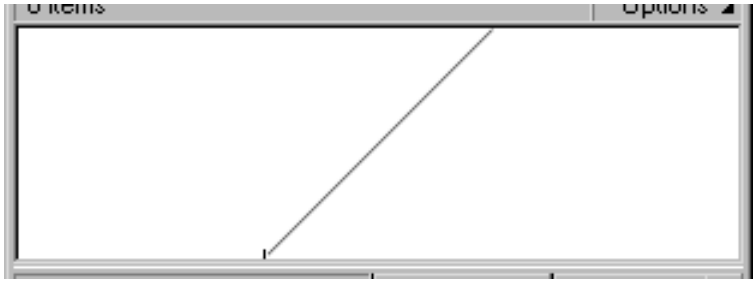
Alle Elemente werden in einem Movieclip erstellt, der sich im ersten und einzigen Frame auf der Hauptbühne befindet.

Zunächst brauchen wir einen Hintergrund, damit sich die Navigation abhebt. Maße und Farben können wir an dieser Stelle nicht verbindlich empfehlen; die Farbe, die wir gewählt haben, ist hellgrün, die Maße sind 100 x 500 Pixel.

Dann muss ein Kreis (3D-Effekt durch radialen Farbverlauf) in einem Movieclip angelegt werden. Der vollständige Kreis soll einen Durchmesser von 17 Pixeln haben. Die Instanz heißt „Kreis“.

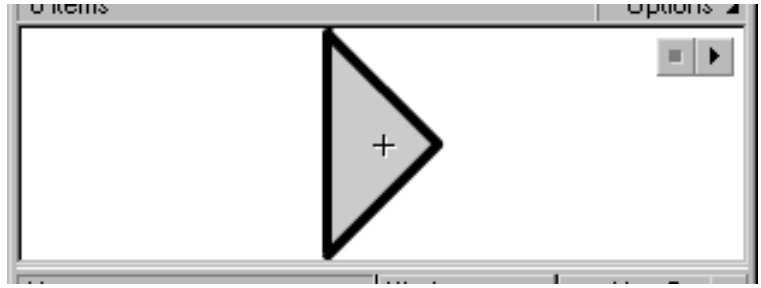


Anschließend ist eine Linie als Movieclip anzulegen. Die Linie verläuft vom Movieclipmittelpunkt im 45°-Winkel nach rechts oben und wird mit den Maßen 100 x 100 Pixel dargestellt.

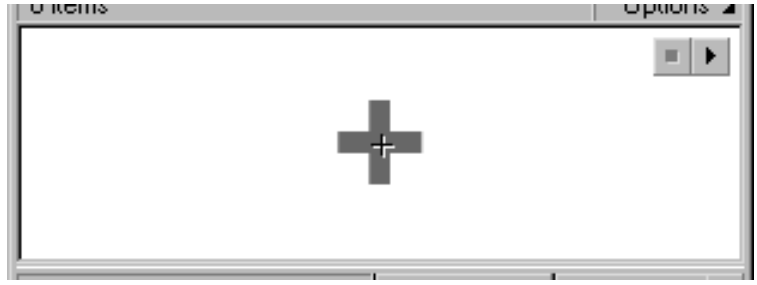


Des Weiteren werden nun einige Buttons benötigt. Den „Kontaktbutton“ erstellen Sie wie gewohnt. Der Inhalt *www.DieFlasher.de* soll auf die Autoren des Flashfilms verweisen.

Wir brauchen jetzt noch weitere Buttons, um die Rotation zu kontrollieren. Wir werden später die Möglichkeit haben, den Würfel im Ursprung zu verschieben, sowie Rotationen um die drei Achsen auszuführen. Um den Würfel zu verschieben, bieten sich Pfeile an. Da Flash es erlaubt, ein Objekt an mehreren Stellen gleichzeitig zu benutzen, soll uns ein Pfeil-Button genügen. Dieser kann wie folgt aussehen:



Des Weiteren „navigieren“ wir den Würfel noch um drei Achsen. Da wir die Rotation mittels der Winkelgeschwindigkeiten kontrollieren, bietet es sich an, + und – zu nehmen, da wir auch „negativ“, sprich in beide Richtungen, drehen können. Dafür werden jetzt zwei Buttons erstellt und in den Movieclip des ersten Frames der Hauptbühne gelegt. Die Buttons können wie folgt aussehen:



Danach brauchen wir noch drei Textfelder zur Ausgabe der Winkelgeschwindigkeiten. Ziehen Sie dafür ein kleines Textfeld mit der Option „Dynamischer Text“ auf. Kopieren Sie es und fügen es noch zweimal ein. Benennen Sie die Felder logisch, d. h. für jeden verständlich. In diesem Fall heißen die drei Textfelder „xachse“, „yachse“, „zachse“.

Nun ordnen Sie die Navigationselemente an. Der „Pfeil“-Button muss noch dreimal eingefügt werden, um eine Ursprungsverschiebung auf der X-Y-Ebene zu realisieren. Das Ergebnis sieht dann wie in der Maginalspalte aus:

Nun fügen Sie in die Clip-Action des Movieclips, in dem alle anderen Elemente sind, zunächst folgende Initialisierungsdaten ein:

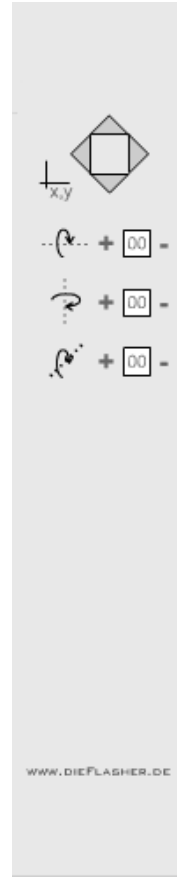
```
// Diese Aktion, also alles innerhalb der {}-Klammern wird einmalig ausgeführt.
onClipEvent (load) {
    // Hier werden die Punktkoordinaten initialisiert
    px0 = 70;
    py0 = 70;
    pz0 = 70;

    px1 = 70;
    py1 = 70;
    pz1 = -70;

    px2 = -70;
    py2 = 70;
    pz2 = -70;

    px3 = -70;
    py3 = 70;
    pz3 = 70;

    px4 = -70;
    py4 = -70;
    pz4 = 70;
```



Auf der CD finden Sie auch eine Beispieldatei, die durchgehend die neue Syntax verwendet.

Ordner 3DWuerfe, Datei: 3DFlash5.fla

```
px5 = 70;
py5 = -70;
pz5 = 70;

px6 = 70;
py6 = -70;
pz6 = -70;

px7 = -70;
py7 = -70;
pz7 = -70;
// Hier wird die maximale Punktanzahl festgelegt: sieben,
weil Null mitgezählt wird.
maxp = 7;
// i, j , k sind die anfänglichen Winkelgeschwindigkeiten.
Wenn Sie den Film ausführen, steht der Würfel.
i = 0;
j = 0;
k = 0;
// Hier wird der X-Y-Ursprung des Würfels festgelegt.
xoffset = 300;
yoffset = 225;
// Die Stärke der Perspektive.
perspect = 5;
// Die Maximalgröße der Punkte.
zoom = 100;
// Eine Geschwindigkeitskontrolle, kann individuell ange-
passt werden, je nach Anzahl der zu berechnenden Punkten.
pitchx = 30;
pitchy = 30;
pitchz = 30;
// Der Weg, die acht Punkte mit den zwölf Linien zu verbin-
den.
knoten = new Array();
knoten[0]=5;
knoten[1]=6;
knoten[2]=7;
knoten[3]=4;
knoten[4]=5;
knoten[5]=6;
knoten[6]=7;
```

```
        knoten[7]=4;
        knoten[8]=1;
        knoten[9]=2;
        knoten[10]=3;
        knoten[11]=0;
        // Entfernung in Z. Kann optional geädert werden.
        entf=-800;
        // Hier werden die acht Eckpunkte des Würfels generiert.
        for (c=0; c<maxp+1; c++) {
            duplicateMovieClip (kreis, "kreis"+c, c);
        }
        // Hier werden die zwölf Verbindungslinien generiert.
        for (lin=0; lin<12; lin++) {
            duplicateMovieClip (_level0.line, "line"+lin, lin+50);
        }
    }
```

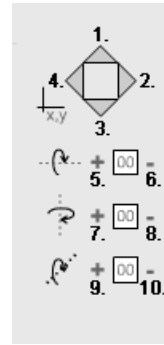
Jetzt, wo die Daten initialisiert wurden, kann man zum zweiten Part wechseln. Dieser wird eingeleitet durch ein

```
    onClipEvent (enterFrame) {
        // Hier wird xr, yr, zr (X, Y, Z - Achse Rotation) initiali-
        siert.
        xr = i/pitchx;
        yr = j/pitchy;
        zr = k/pitchz;
        // Die Funktion calc (Calculate = berechne) wird definiert.
    Übergeben wird ein Wert.
        function calc (wert) {
            // Folgend wird mittels Polarkoordinaten die Punkttrans-
            formation im Raum berechnet.
            eval("zn" add wert) = Math.sin(xr)*eval("py" add
            wert)+Math.cos(xr)*eval("pz" add wert);
            eval("py" add wert) = Math.cos(xr)*eval("py" add wert)-
            Math.sin(xr)*eval("pz" add wert);
            eval("pz" add wert) = eval("zn" add wert);
            eval("zn" add wert) = Math.cos(yr)*eval("pz" add wert)-
            Math.sin(yr)*eval("px" add wert);
            eval("px" add wert) = Math.cos(yr)*eval("px" add
            wert)+Math.sin(yr)*eval("pz" add wert);
            eval("pz" add wert) = eval("zn" add wert);
```

```
            eval("yn" add wert) = Math.cos(zr)*eval("py" add wert)-
            Math.sin(zr)*eval("px" add wert);
            eval("px" add wert) = Math.cos(zr)*eval("px" add
            wert)+Math.sin(zr)*eval("py" add wert);
            eval("py" add wert) = eval("yn" add wert);
            // Hier wird mittels pz die „Tiefe" des Punktes im Raum
            gemessen. Um den Punkt in Fluchtpunktnähe zu verschie-
            ben, wird hier ein Korrekturfaktor berechnet, der die
            gleich zugeordneten Werte auf der X-Y-Ebene mehr Rich-
            tung Mitte verschiebt.
            eval("deep" add wert)=entf/(entf-eval("pz" add wert));
// Hier werden die Punkte selbst skaliert, ihre Größe der
„Tiefe", also Entfernung zum Betrachter entsprechend verklei-
nert.
            setProperty ("kreis" add wert, _xscale, zoom-eval("pz"
            add wert)/perspect);
            setProperty ("kreis" add wert, _yscale, zoom-eval("pz"
            add wert)/perspect);
            // Hier werden die Koordinaten dem Punkt zugeordnet.
            setProperty ("kreis" add wert, _x, eval("px" add
            wert)*eval("deep" add wert)+xoffset);
            setProperty ("kreis" add wert, _y, eval("py" add
            wert)*eval("deep" add wert)+yoffset);
            setProperty ("kreis" add wert, _alpha, 40);
        }
        // Hier wird die Berechnung jedes einzelnen Punktes aufgeru-
        fen.
        for (c=0; c<maxp+1; c++) {
            calc(String(c));
        }
        // Folgend werden die Verbindungslinien zwischen den einzel-
        nen Punkten berechnet.
        for (lin=0; lin<12; lin++) {
            li=knoten[lin];
            kn=lin;
           // Durch das „kn %= 8" wird kn automatisch für Werte grö-
           ßer als Acht um Acht subtrahiert.
            kn %= 8;
            // Die Position der Linie wird an die Position des Punk-
            tes angeglichen.
            eval("line"+lin)._x=eval("kreis" + kn)._x;
            eval("line"+lin)._y=eval("kreis" + kn)._y;
```

```
            // Da die Linie die Maße 100 x 100 Pixel hat, hat
            _xscale, bzw. _yscale gleichzeitig die realen Pixel-
            werte. Es werden durch Berechnen des Abstandes die Stre-
            ckungs- bzw. Stauchungsfaktoren berechnet.
            eval("line"+lin)._xscale=-(eval("kreis" + kn)._x-
                eval("kreis" + li)._x);
            eval("line"+lin)._yscale=eval("kreis" + kn)._y-
                eval("kreis" + li)._y;
        }
    }
```

Jetzt haben Sie (hoffentlich) alle Scripts richtig eingefügt, die Instanznamen richtig vergeben und die Filme in der richtigen Hierarchie eingefügt. Nun muss der Navigation noch Leben eingehaucht werden. Im Folgenden numerieren wir die Elemente durch, das jeweils dazugehörige ActionScript werden Sie darunter finden.



Folgend das dazu gehörige ActionScript:

❶ on (press) {
yoffset=yoffset-5;
}
	❷ on (press) {
	xoffset=xoffset+5;
	}

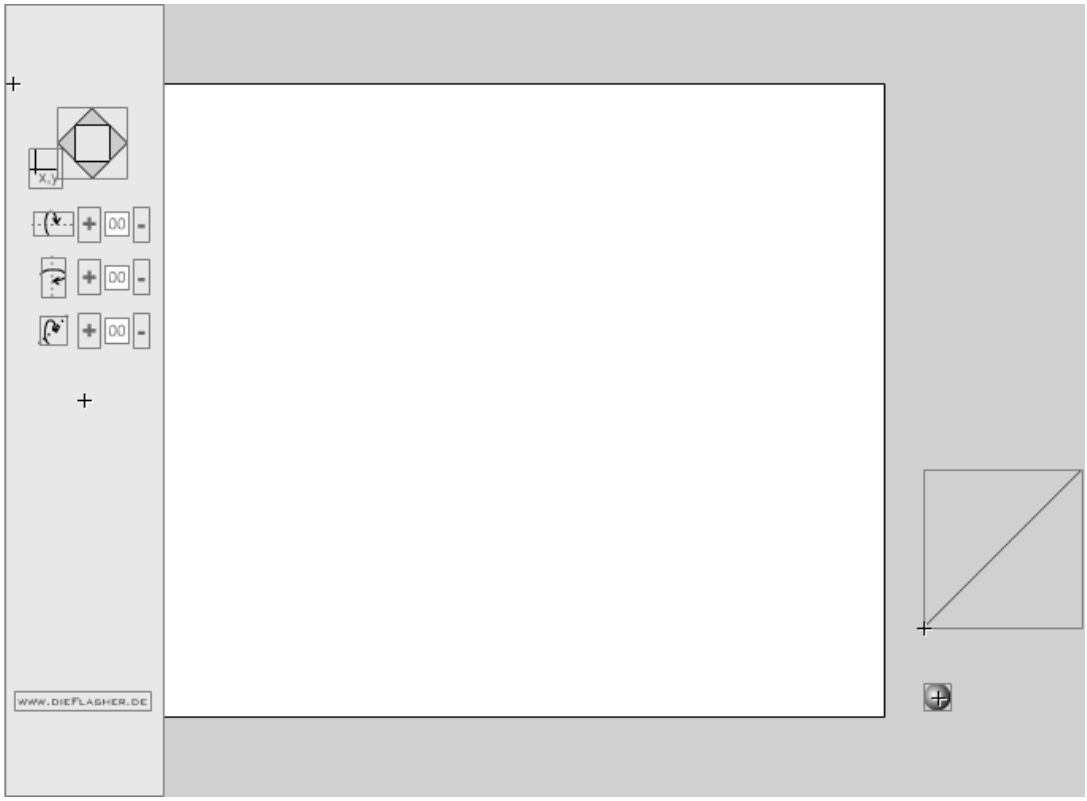
❸ on (press) {
yoffset=yoffset+5;
}
	❹ on (press) {
	xoffset=xoffset-5;
	}

❺ on(press){
i=i+1;
xachse=i;
}
	❻ on(press){
	i=i-1;
	xachse=i;
	}

❼ on (press){
j=j+1;
yachse=j;
}
	❽ on (press){
	j=j-1;
	yachse=j;
	}

❾ on (press){
k=k+1;
zachse=k;
}
	❿ on (press){
	k=k-1;
	zachse=k;
	}

Wenn alle Elemente richtig platziert wurden, darf die Hauptbühne nur einen MovieClip beinhalten, der aber die komplette Berechnung per ActionScript in sich trägt. Das sieht auf der Hauptbühne dann wie folgt aus:



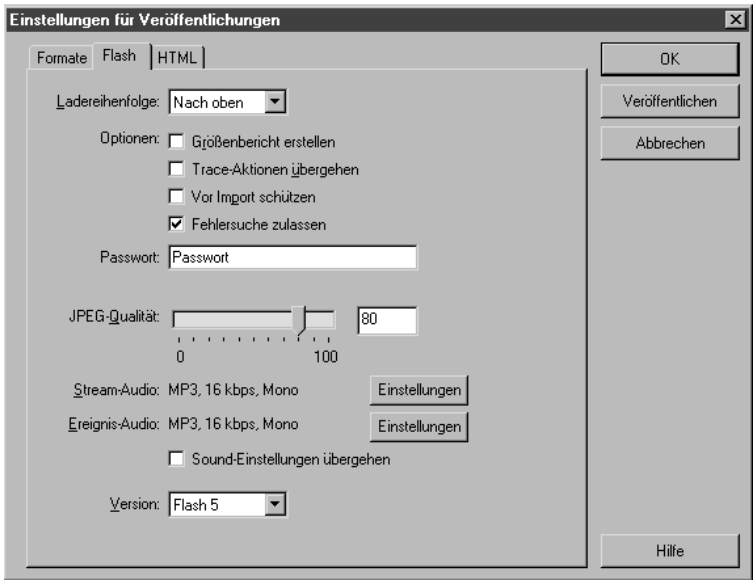
Cinema 4D hat eine Koordinatenangabe, die hier Verwendung finden kann.

Wenn Sie überlegen, an Stelle des Würfels z. B. eine Pyramide oder ein aus Punkten zusammengesetztes Objekt zu „importieren“, so können Sie dies. Nur müssen Sie den `Array()` wegen der Linien etc. manuell anpassen. Ansonsten fügen Sie einfach fortlaufend die Punkte ein, ändern zu guter Letzt noch *maxp,* die Variable, in der die Anzahl der Punkte im Raum gespeichert ist.

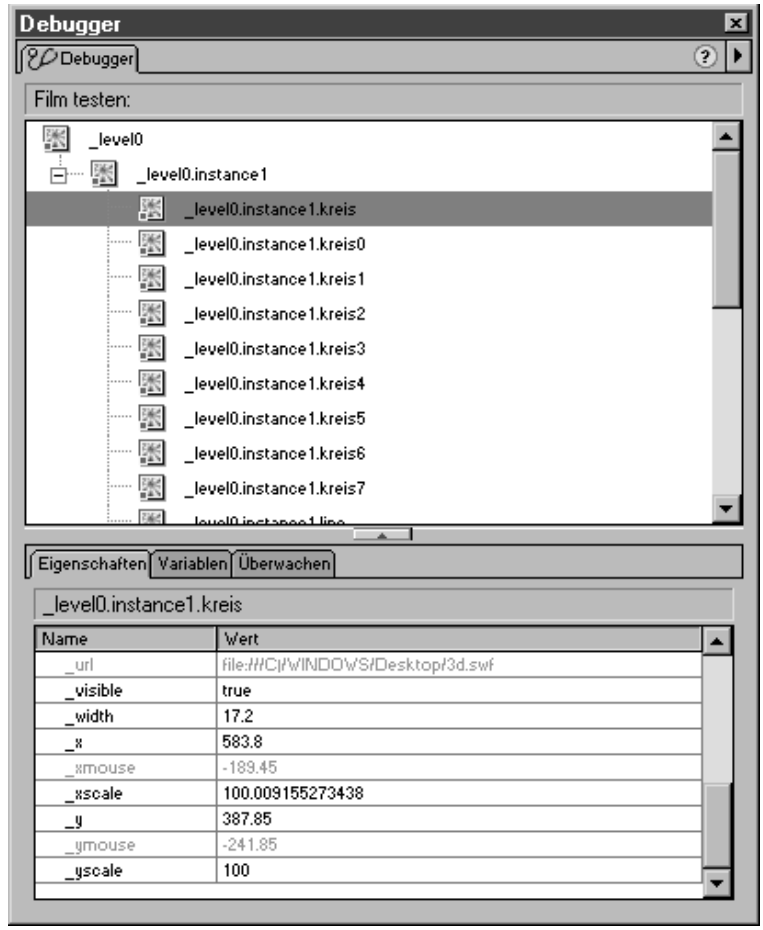
Diese Engine halten wir für durchaus erweiterbar, sie kann nun individuellen Kundenwünschen angepasst werden. Die Koordinatensteuerung kann z. B. alternativ per Mausbewegung abgefragt werden und die Punkte in Buttons verwandelt werden.

7 Der Debugger

Auch wenn Sie denken, bisher könnten keine Probleme auftauchen, so möchte ich Ihnen doch den Umgang mit diesem neuen Werkzeug ans Herz legen. Umsteiger werden der Alternative, dem Flash-4-Befehl `trace()` nicht nachtrauern müssen. Er wird Ihnen bei der weiteren Fehlersuche behilflich sein. Um den Debugger zu aktivieren, müssen Sie bei den Einstellungen für Veröffentlichungen Strg + ⇧ + F12 „Fehlersuche zulassen“ anwählen sowie ein Passwort eingeben.



Wenn Sie den Film mittels Strg + ↵ in der Vorschau anschauen, kommen Sie per rechtem Mausklick auf den Film in ein erweitertes Kontextmenü.
Wenn Sie DEBUGGER ausgewählt haben, öffnet sich ein neues Dialogfeld.



Hier haben Sie die Möglichkeit, Eigenschaften der Instanzen sowie Werte der benutzten Variablen abzurufen und während des Abspielens zu kontrollieren.

8 Flash Workarounds

Wir beschreiben in dem folgenden Kapitel Lösungen und Alternativen, um bekannte Probleme, die Ihnen in Flash 4 und Flash 5 begegnen, zu umgehen.

8.1 Sinus/Kosinus (Flash 4)

Die bekannten Winkelfunktionen werden nicht standardmäßig von Flash 4 unterstützt, hierbei greift man am besten auf das Taylor-Polynom zurück, das aus der Mathematik bekannt ist. Es bietet die Möglichkeit, jede stetig differenzierbare Funktion beliebig genau darzustellen. Der programmiertechnisch interessante Aspekt ist, dass diese Funktion nur über einem symmetrischen Intervall genau ist. Hängt man die Funktion z.B. in 0 auf, so ist die Näherung zur 0 hin am genauesten. Von daher sollte darauf geachtet werden, dass keine Winkel berechnet werden, die größer als +/- 360° sind.

Das folgende Script ist Flash-4-kompatibel und aufgrund der hohen Anforderung an die Rechenleistung des Prozessors auch nicht für komplexe Anwendungen wie 3D-Engines geeignet.

Sinus()

Hier ist „grad“ der Winkel, der berechnet werden soll. „y“ wird am Ende den Wert von sin(grad) haben, so dass gilt:

```
// y = sin (grad)
// Die Variablen werden initialisiert:
Set Variable: "Pi" = 3.14159265358979
Set Variable: "x" = 2*Pi*grad/360
Set Variable: "iteration" = 20
Set Variable: "iteration" = iteration*2
Set Variable: "xpotenz" = x
Set Variable: "y" = 0
Set Variable: "i" = 1
Set Variable: "f" = 1
```

Sie können dies in Flash 5 auch in Form einer Funktion deklarieren.

Plusminus wird im Folgenden gebraucht, da das Taylorpolynom auf einer Summe von Ableitungen in einem Punkt basiert. Der Einfachheit halber nimmt man o. Da fortlaufend sin() nach cos() nach -sin() nach -cos() nach sin() abgeleitet wird, lässt sich dieser Ausdruck mittels einer Schleife realisieren:

```
Set Variable: "plusminus" = 1
```

Die Anzahl der Iterationen bestimmt die Genauigkeit des Ergebnisses:

```
Loop While (i<iteration)
      Set Variable: "y" = y + (plusminus*(xpotenz/f))
      Set Variable: "f" = f*(i+1)*(i+2)
      Set Variable: "i" = i + 2
      Set Variable: "xpotenz" = xpotenz*x*x
      Set Variable: "plusminus" = plusminus * (-1)
End Loop
```

Mittels dieser Abfrage werden Übertragungsfehler korrigiert:

```
If ((y<0.0000000001) AND (y>-0.0000000001))
      Set Variable: "y" = 0
End If
```

Kosinus()

Hier ist „grad“ der zu berechnende Winkel. „z“ wird am Ende den Wert von cos(grad) haben, so dass gilt:

z = cos(grad)

Abgesehen davon, dass Sie „z“ = o anstelle von „y“ = o initialisieren, können Sie obige Initialisierung wiederverwenden:

```
Loop While (i<iteration)
      Set Variable: "z" = z + (plusminus*(xpotenz/f))
```

Da der Sinus eine Ableitung des Kosinus ist, reicht es, mit f • (i) • (i + 1) anstelle von f • (i + 1) • (i + 2) zu rechnen:

```
      Set Variable: "f" = f*(i)*(i+1)
      Set Variable: "i" = i + 2
      Set Variable: "xpotenz" = xpotenz*x*x
      Set Variable: "plusminus" = plusminus * (-1)
End Loop
```

Mittels dieser Abfrage werden Übertragungsfehler korrigiert:

```
If ((z<0.0000000001) AND (z>-0.0000000001))
      Set Variable: "z" = 0
End If
```

8.2 Rotationsbug (Flash 4)

Dreht man eine Instanz per ActionScript-Befehl

```
setProperty ("/Ziel", _rotation, Wert);
```

so verliert die Instanz, sofern es sich um eine fortlaufende Rotation handelt, stetig an Größe. Dies kann man durch den Einschub der Zeilen

```
setProperty ("/Ziel ", _xscale, 100);
setProperty ("/Ziel ", _yscale, 100);
```

verhindern.

8.3 Bildqualität-Bug (Flash 4)

Legt man auf der Hauptbühne mehrere Frames an, erhält man einen Bildqualitätsverlust. Wird der Film so angelegt, dass alles in Instanzen läuft und auf der Hauptbühne nur ein Frame existiert, werden die Bilder wieder scharf (zu beobachten wenn man versucht, Screenshots einzubinden, bei denen die Schrift lesbar bleiben muss, z. B. für eine interaktive CD-ROM). Wird dieser Film aus einem anderen Film geladen, der mehr als ein Frame auf der Hauptbühne enthält, werden die Bilder wieder unscharf.

8.4 Fehler beim Widerrufen von Befehlen

Der Button, um den letzten Befehl rückgängig zu machen, wurde bei Flash 5 so angelegt, dass dieser für das Actionscriptfenster und die Hauptbühne separat agiert, je nachdem, welches Fenster gerade ausgewählt ist. Leider wird beim Drücken des Buttons automatisch die Hauptbühne angewählt, so dass ein Druck auf den Button stets den letzten Befehl auf der Hauptbühne widerruft.

Um einen Befehl im ActionScriptfenster rückgängig zu machen, verwenden Sie die Tastenkombination Strg + z.

8.5 Fehler bei der Vergabe von Framebezeichnungen oder Instanznamen

Wenn Sie in das jeweilige Textfeld einen Namen für einen Frame oder eine Instanz eingeben und anschließend ein anderes Fenster aufrufen oder auf die Hauptbühne klicken, wird dieser Name nicht gespeichert. Damit Flash diesen Namen behält, muss man noch einmal zur Bestätigung in dasselbe Fenster klicken, um das Textfeld zu verlassen. Erst dann speichert Flash den Namen.

Es ist nicht möglich, eingeladene Movies zu duplizieren.

8.6 Tipps / Wissenwertes

_xmouse und _ymouse

Die Mauskoordinaten werden meistens mit `_xmouse` und `_ymouse` aus einem Movieclip abgefragt. Erst durch die Eingabe von `_root._xmouse` werden die Koordinaten wieder relativ zur Hauptbühne übermittelt.

Variablen

Variablen müssen mit einem Buchstaben beginnen und dürfen keine Sonderzeichen wie Umlaute etc. enthalten.

Verwenden Sie nicht die gleichen Namen für Instanzen, Textfelder, Objekte, Funktionen oder Variablen, wenn diese in demselben Movie liegen. Grundsätzlich sollte man unterschiedliche Namen verwenden.

If-Abfragen

Bei einer If-Abfrage gibt Flash keine Fehlermeldung aus, falls man fälschlicherweise nur ein Gleichzeichen benutzt.

LoadMovie()

Bei `loadMovie` kann man nicht die neu eingeführte Punktsyntax benutzen, um Variablen aus einem anderen CGI zu übergeben.

```
instanzname.loadMovie(xxx);
```

wäre also nicht möglich.

OnClipEvent(load)

Die Befehle im `onClipEvent(load)` und die Befehle im Frame des Movies sind kumulativ: Wenn z.B. bei einem `OnClipEvent(load)` der Befehl `GotoAndPlay(2);` steht und im ersten Frame des entsprechenden Movies ein `Stop();`-Befehl, springt das Movie zu Frame 2 und stoppt, anstatt wie im `onClipEvent()` beschrieben, dieses abzuspielen.

Duplizieren von Movies

Es ist nicht möglich, eingeladene Movies zu duplizieren. Statt dessen wird das Movie dupliziert, in das das SWF eingeladen wurde.

„Verwenden Sie die Tags OBJECT und EMBED zum Anzeigen von Flash“

Wenn Sie diesen Schriftzug nicht immer in Ihrem erstellen HTML-Text haben möchten, können Sie diesen einmal aus der Default.html im Flash-5-Installationsverzeichnis (Standard Macromedia/Flash5) unter HTML löschen. Die Default.html sollte dann anschliessend so aussehen:

```
$TTNur Flash (Standard)
$DS$DF
<HTML DIR=LTR>
<HEAD>
<TITLE>$TI</TITLE>
</HEAD>
<BODY bgcolor="$BG">

<!-- URL's used in the movie--> $MU <!-- text used in the movie-
-> $MT

<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
 codebase="http://download.macromedia.com/pub/shockwave/cabs/
flash/swflash.cab#version=5,0,0,0"
 WIDTH=$WI HEIGHT=$HE>
 $PO <EMBED $PE WIDTH=$WI HEIGHT=$HE
 TYPE="application/x-shockwave-flash" PLUGINSPAGE="http://
www.macromedia.com/shockwave/download/
index.cgi?P1_Prod_Version=ShockwaveFlash"></EMBED>
</OBJECT>

</BODY>
</HTML>
```

Unbekannter Projektor-Befehl

Wenn Sie einen Flash-Film in Form des Projektors (.exe) veröffentlichen, bietet sich Ihnen die Möglichkeit, Variablen in externe Dateien abzuspeichern. Möchten Sie z.B. Einstellungen einer interaktiven CD-ROM auf dem entsprechenden Anwenderrechner ablegen, so ist dies über den folgenden Befehl möglich:

Falls Sie dies ausprobieren, so beachten Sie die Zugriffsrechte.

```
FSCommand(„save","laufwerk:/dateiname");
```

Es werden alle verwendeten Variablen in die angegebene Datei gespeichert. Dies bietet natürlich auch die Möglichkeit des Missbrauches, aber da es sich bei dem Dateiformat um ein Executable (.exe) handelt, geht von dieser Funktion wohl ein relativ geringes Risiko aus.

Pearson Education

ADDISON-WESLEY

9 ActionScript-Referenzteil

Im folgenden Kapitel finden Sie alphabetisch geordnet alle Flash-ActionScript-Befehle.

A

Abs (Flash 5)
→ Math.Abs().

Acos (Flash5)
→ Math.acos().

Add (Flash 4)
Dient als Bindeglied zwischen Variablen.

```
var1 = "Hello ";
var2 = var1 add " World";
```

var2 enthält nun die Zeichenkette „Hello World“.

_alpha (Flash 4)

```
var1 = instanzname._alpha;
```

Setzt bzw. liest die Transparenz einer Instanz.

```
var1 = _root.InstanzName._alpha
var1 = var1 / 2;
_root.InstanzName._alpha = var1;
```

Die Transparenz der Instanz mit dem Namen *Instanzname* wurde um 50% erhöht bzw. die Sichtbarkeit um 50% gesenkt.

And (Flash 4)
Logisches „Und“ kann in Flash 5 auch durch && ersetzt werden.

`Wert1 && Wert2` oder `Wert1 and Wert2`

Wert 1	Wert 2	Ergebnis
true	true	true
true	false	false
false	true	false
false	false	false

Array()

Array
→ *6.1 Memory*
– Das Duplizieren der Memorykarten.

```
Datensatz = new Array();
```

Legt einen Datensatz in Form eines Arrays bzw. Vektors an. Es können beliebige Werte zugeordnet werden.

```
Datensatz = new Array();
Datensatz[0] = Wert;
...
Datensatz[n] = Wert;
```

Der Datensatz startet mit dem Index 0. Weiter wird fortlaufend in ganzen natürlichen Zahlen indiziert. Als Wert ist String und Zahl zulässig.

Auch ein mehrdimensionales Array kann erstellt werden.

```
name = new Array();
name[0] = new
Array("feld00","feld01","feld02");
name[1] = new
Array("feld10","feld11","feld12","feld13");
```

```
name[2] = new Array("feld20","feld21");
name[3] = new
Array("feld30","feld31","feld32");

for (i=0; i< name.length; i++) {
for (i2=0; i2< name[i].length; i2++) {
trace ("name"+i+""+i2+" = "+ name[i][i2]);
}
}
```

Der trace-Befehl würde nur die Werte des Arrays ausgeben, die auch gesetzt wurden, da die for-Schleifen nur über die existierenden Felder laufen. Dies wird mit `length` überprüft.

Die Anführungszeichen beim Setzen der Arrays stehen dort nur, um direkt Strings in die Arrays zu setzen. Ohne Anführungszeichen würde `feld21` als Variable deklariert. Für zweidimensionale Arrays ist diese Schreibweise noch brauchbar, wenn auch umständlich.

Man kann ein mehrdimensionales Array leicht mit einigen verschachtelten for-Schleifen erzeugen.

```
name = new Array();
for (i=0; i<10; i++) {
  name[i] = new Array();
  for (i2=0; i2<10; i2++) {
    name[i][i2] = new Array();
  }
}
name[1][2][3] = "hallo";
name[6][3][4] = "welt";
for (i=0; i<name.length; i++) {
  for (i2=0; i2<name[i].length; i2++) {
    for (i3=0; i3<name[i][i2].length; i3++)
{
      trace ("name"+i+""+i2+""+i3+" =
                          "+name[i][i2][i3]);
    }
  }
}
```

Zuerst wird ein dreidimensionales Array erzeugt. Die erste for-Schleife greift dabei nacheinander auf die Elemente des ersten Arrays zu und erstellt in diesem ein weiteres Array.

Das gleiche macht die zweite for-Schleife: Diese greift auf die Elemente des zweiten Arrays zu, welche im ersten Array liegen, und erstellt darin wieder ein neues Array.

Das erstellte dreidimensionale Array wäre 10 x 10 x 10 Felder groß. Da das erste Array nicht begrenzt ist, wäre es theoretisch unendlich groß. Man braucht aber ein dreidimensionales Array, und dieses liegt nur in den ersten 10 Feldern, so dass wir dies vernachlässigen.

Wenn man ein größeres Array erzeugen muss, kann man die Werte der Schleifendurchläufe erhöhen. Diesen Lösungsansatz kann man beliebig erweitern und so ohne Probleme fünf- oder zehndimensionale Arrays erzeugen.

In dem Beispiel werden anschließend als Beweis noch zwei beliebige Felder, hier [1][2][3] und [6][3][4], mit einem String versehen. Diese werden dann mittels `trace` ausgegeben.

Array.concat

```
Datensatz.concat(newDatensatz);
```

Verkettet zwei Datensätze.

```
Datensatz0 = new Array (1,2,3);
Datensatz1 = new Array (4,5,6);
Datensatz2 = Datensatz0.concat(Datensatz1);
```

Der Inhalt von Datensatz 2 entspricht nun [1, 2, 3, 4, 5, 6]. Die Inhalte der Datensätze können heterogen sein, es können verschiedene Datentypen kombiniert werden.

Array.join()

```
Datensatz = new Array()
Var = Datensatz.join(String);
```

Zerlegt eine Zeichenkette (Var) in einzelne Elemente, die durch bestimmte Zeichen getrennt sind (String), und ordnet diese dem entsprechenden Datensatz zu. Aus `Var = „links $ mitte $ rechts"` wird so mittels `String = „ $"` der Array `Datensatz = [„links",„mitte",„rechts"]`.

Array.length

```
Datensatz.length;
```

Gibt die Anzahl der Datensatzeinträge an, mit o beginnend.

Array.pop

```
Datensatz.pop();
```

Löscht den letzten Eintrag des Datensatzes.

Array.push

```
Datensatz.push(Wert1, Wert2, ...)
```

Fügt `Wert1, Wert2, ...` am Ende des Datensatzes hinzu.

Array.reverse

```
Datensatz.reverse();
```

Kehrt die Reihenfolge der Datensatzeinträge um.

Array.shift

```
Datensatz.shift()
```

Löscht den ersten Eintrag des Datensatzes.

Array.slice

```
Datensatz.slice(start, ende);
```

Kopiert einen Teil des Datensatz von Start bis Ende.

```
A = new Array;
B = new Array;
A[0]="a";
A[1]="b";
A[2]="c";
A[3]="d";
A[4]="e";
A[5]="f";
A[6]="g";
B=A.slice(2,5);
```

Der Datensatz B nimmt die Werte [„c", „d", „e", „f"] an, A bleibt dabei unverändert.

Array.sort

```
Array.sort(Sortierkriterium);
```

Sortiert ein Array z. B. alphabetisch.

Array.splice

```
Datensatz.splice(start, löschnum, Wert,
Wert1, ...);
```

Löscht im Datensatz von der Position `start` an Einträge und schreibt an deren Stelle die ggf. vorhandenen Werte: `Wert, Wert1, ...,Wertn`.

Array.toString

```
Zeichenkette = Datensatz.toString;
```

Es werden alle Daten aus dem Datensatz in einen String gereiht. Wenn die Einträge `Datensatz[0] = „a", Datensatz[1] = „b", Datensatz[2] = „c"` lauten, so lautet die erzeugte Zeichenkette `„a, b, c"`.

Asin (Flash 5)

→ Math.asin().

Atan (Flash 5)

→ Math.atan().

attachMovie

→ MovieClip.attachMovie.

B

Boolean

```
Boolean(Ausdruck);
```

Boolean überprüft den Ausdruck und nimmt entsprechend `true` oder `false` an.

Boolean() (Flash 5)

```
Boolean(Ausdruck);
```

Gibt den Wahrheitswert einer Aussage zurück. So liefert der Ausdruck:

```
Boolean(„Bock"=="Gärtner")
```

false zurück.

Boolean(Objekt)

```
Bool = new Boolean;
```

Nun können Sie Bool `true` oder `false` zuordnen.

Boolean.toString

```
Boolean.toString();
```

Hier wird `true` oder `false` als „true" bzw. „false" ausgegeben. Der logische Ausdruck wird in eine Zeichenkette verwandelt.

break (Flash 4)
Unterbricht folgende Schleifentypen:

- for
- for ... in
- do ... while
- while

Die jeweilige Schleife wird bei diesem Befehlsaufruf sofort terminiert. Die nachfolgenden Befehle außerhalb der Schleife werden anschließend ausgeführt.

C

Call() (Flash 4)
Führt das ActionScript des aufgerufenen Ziellabels aus. Mit Hilfe des Befehls lassen sich Funktionsaufrufe in Flash 4 umsetzen. Anstatt also Funktionen zu erstellt, erstellt man ein Frame und gibt diesem einen Labelnamen. Mit `call(Labelnamen)` kann man dann das in diesem Frame enthaltene ActionScript ausführen und erhält so auch eine Funktion.

Chr() (Flash 4)
Wandelt einen ASCII-Wert in das entsprechende Zeichen um. → *String.fromCharCode*

Color (Objekt)

Colorobjekt,
→ *6.1 Memory – Die Farbe der Karte.*

Mit dem Color-Objekt können Sie auf die Farbwerte eines Movieclips zugreifen. Sie können diese setzen sowie auch lesen.

```
ColorObjektName = new color(_root.Instanz-
name);
```

Ein Colorobjekt wird erstellt, um mit den Funktionen `setRGB`, `getTransform` usw. auf die Eigenschaften des Movies, auf das das Colorobjekt zeigt, zugreifen zu können.

Color.getRGB
Übermittelt die RGB-Werte eines Movies.

```
Var = ColorObjektName.getRGB;
```

Der Farbwerte sind nun in der Variable `Var` gespeichert.

Die RGB-Werte kann man erst abfragen, nachdem man diese mit `setRGB` gesetzt hat, ansonsten ist dieser Wert o.

Color.getTransform
Übermittelt die Farbwerte eines Movies.

```
Var = new object;
Var = ColorObjektName.getTransform
```

Die Farbwerte kann man erst abfragen, nachdem man diese mit `setTransform` gesetzt hat.

Color.setRGB

→ *6.1 Memory – Die Farbe der Karte.*
Ändert die RGB-Werte eines Movies. Die RGB-Werte müssen mit oxRRGGBB angegeben werden.

```
ColorObjectName.setRGB(RGB-Werte);
```

Befehl	Aktion
setRGB	setzt die RGB-Farbwerte eines Movies, auf dass das Color-Objekt ausgerichtet ist.
setTransform	setzt die Farbtransformationswerte für ein Color-Objekt.
getRGB	ermittelt die RGB-Werte eines Movieclips, falls dieses zuvor gesetzt wurden.
getTransform	ermittelt die Transformationsinformationen eines Movieclips, falls diese zuvor gesetzt worden sind.

Color.setTransform

Ein Objekt, mit dem man direkt auf alle Farbwerte eines Movies zugreifen kann. Im nachfolgenden Beispiel kann man die Farbwerte `Rot`, `Rotffset` usw. entweder als Variablen ansehen und diesen zuvor Werte zuordnen oder direkt gegen Werte tauschen. Die Werte der Farben müssen zwischen -100 und 100 liegen, der Offsetbereich geht von -255 bis 255. Zusätzlich kann man auf diesem Weg auch die Transparenz (Alpha) regulieren.

```
ColorObjekt = new color(_root.Instanzname);
  TransformObjekt = new object();
  TransformObjekt.ra = Rot;
  TransformObjekt.rb = RotOffset;
  TransformObjekt.ga = Gruen;
  TransformObjekt.gb = GruenOffset;
  TransformObjekt.ba = Blau;
  TransformObjekt.bb = BlauOffset;
  TransformObjekt.aa = Alpha;
  TransformObjekt.ab = AlphaOffset;
  ColorObjekt.setTransform(TransformObjekt);
```

Concat

→ Array.concat.

continue (Flash 4)

```
...
continue;
...
```

Weist Flash an, innerhalb einer Schleife (while / do... while / for / for...in) die folgenden Befehle zu überspringen und zum Schleifenanfang zurückzuspringen.

_currentframe

```
Instanzname._currentframe
```

Gibt die aktuelle Abspielposition (Frame) der gewählten Instanz an.

D

Date() (Flash 5)

Dateobjekt,
→ *6.1 Memory*
– Anzeige der Uhrzeit.

Mit dem Date-Objekt ist es möglich, verschiedene Zeiten und Daten abzurufen. Sie können die lokale Zeit abrufen oder die Weltzeit (GMT, heute UTC). Die lokale Zeit wird durch das Betriebssystem bestimmt, auf dem der FlashPlayer ausgeführt wird.

Um das Date-Objekt nutzen zu können, müssen Sie es zuerst erzeugen

```
Name =New Date();
```

Man kann im übrigen auch Date-Objekte mit Bezug auf eine feste Zeit erstellen.

```
new Date (jahr, monat  , datum  , stunde  ,
minute  , sekunde  , millisekunde );
```

Alle Werte bis auf Jahr sind optional. Optional bedeutet, dass diese Werte auch entfallen können.

Beispiel für ein erstelltes Zeitobjekt:

ZB-)

```
zeit = new date();
sec = zeit.getseconds();
min = zeit.getminutes();
std = zeit.gethours();
Uhrzeit = std+" : "+min+" : "+sec;
```

- Date.getDate

```
var1=name.getDate();
```

Date-Objekt
→ *6.1 Memory*
– Anzeige der Uhrzeit.

Gibt den Tag des Monats des angegebenen Date-Objektes nach lokaler Uhrzeit zurück. Dies ist eine ganze Zahl von 1 bis 31.

- Date.getDay

Anzeige des Datums
→ *114 Gästebuch.*

```
var1=name.getDay();
```

gibt den Tag des Monats des angegebenen Date-Objektes nach lokaler Zeit zurück. 0 für Sonntag, 1 für Montag usw.

- Date.getFullYear

```
var1=name.getFullYear();
```

Gibt die vollständige Jahreszahl des angegebenen Date-Objektes nach lokaler Zeit zurück. Dies ist eine vierstellige Zahl, z.B. 2000.

- Date.getHours

```
var1=name getHours();
```

Gibt die Stunde des angegebenen Date-Objektes nach lokaler Zeit zurück. Dies ist eine ganze Zahl von 0 bis 23.

- Date.getMilliseconds

```
var1=name.getMilliseconds();
```

Gibt die Millisekunden des angegebenen Date-Objektes nach lokaler Zeit zurück. Dies ist eine ganze Zahl von 0 bis 999.

- Date.getMinutes

```
var1=name.getMinutes();
```

Gibt die Minuten des angegebenen Date-Objektes nach lokaler Zeit zurück. Dies ist eine ganze Zahl von 0 bis 59.

- Date.getMonth

```
var1=name.getMonth();
```

Gibt den Monat des angegebenen Date-Objektes nach lokaler Zeit zurück. 0 für Januar, 1 für Februar usw.

- Date.getSeconds

```
var1=name.getSeconds();
```

Gibt die Sekunden des angegebenen Date-Objektes nach lokaler Zeit zurück. Dies ist eine ganze Zahl von 0 bis 59.

- Date.getTime

```
var1=name.getTime();
```

Gibt die Differenz zwischen der lokalen Zeit des Computers und der Weltzeit in Minuten zurück.

- Date.getTimezoneOffset

```
var1=name.getTimezoneOffset();
```

Gibt den Tag (das Datum) des Monats des angegebenen Date-Objektes nach Weltzeit zurück.

- Date.getUTCDate

```
var1=name.getUTCDate();
```

Gibt den Wochentag des angegebenen Date-Objektes nach Weltzeit zurück. Dies ist eine ganze Zahl von 1 bis 31

- Date.getUTCDay

```
var1=name.getUTCDay();
```

Gibt den Wochentag des angegebenen Date-Objektes nach Weltzeit zurück. 0 für Sonntag, 1 für Montag usw.

- Date.getUTCFullYear

```
var1=name.getUTCFullYear();
```

Gibt die vierstellige Jahreszahl des angegebenen Date-Objektes nach Weltzeit zurück. Dies ist eine vierstellige Zahl, z.B. 2000.

- Date.getUTCHours

```
var1=name.getUTCHours();
```

Gibt die Stunden des angegebenen Date-Objektes nach Weltzeit zurück. Dies ist eine ganze Zahl von 0 bis 23.

- Date.getUTCMilliseconds

```
var1=name.getUTCMilliseconds();
```

Gibt die Millisekunden des angegebenen Date-Objektes nach Weltzeit zurück. Dies ist eine ganze Zahl von 0 bis 999.

- Date.getUTCMinutes

```
var1=name.getUTCMinutes();
```

Gibt die Minuten des angegebenen Date-Objektes nach Weltzeit zurück. Dies ist eine ganze Zahl von 0 bis 59.

- Date.getUTCMonth

```
var1=name.getUTCMonth();
```

Gibt den Monat des angegebenen Date-Objektes nach Weltzeit zurück. 0 für Januar, 1 für Februar usw.

- Date.getUTCSeconds

```
var1=name.getUTCSeconds();
```

Gibt die Sekunden des angegebenen Date-Objektes nach Weltzeit zurück. Dies ist eine ganze Zahl von 0 bis 59.

- Date.getYear

```
var1=name.getYear()+1900;
```

Gibt das Jahr des angegebenen Date-Objektes nach lokaler Zeit zurück. Zu dem Datum muss man 1900 addieren, da für das Jahr 2000 100 ausgegeben wird.

- Date.setDate

```
var1=name.setDate(datum);
```

Setzt den Tag des Monats für das angegebene Date-Objekt nach lokaler Zeit.

- Date.setFullYear

```
var1=name.setFullYear(jahr  , monat  ,
datum ));
```

monat: eine Zahl von 0 (Januar) bis 11 (Dezember) (optional)

datum: eine Zahl von 1 bis 31 (optional)

Eine vierstellige Zahl zur Angabe eines Jahres. Zweistellige Zahlen sind nicht für die Jahresangabe vorgesehen. 99 steht dementsprechend nicht für 1999, sondern für das Jahr 99.

- Date.setHours

```
var1=name.setHours(stunde);
```

stunde: eine ganze Zahl von 0 bis 23

Setzt die Stunde des angegebenen Date-Objektes nach lokaler Zeit.

- Date.setMilliseconds

```
var1=name.setMilliseconds(millisekunde);
```

millisekunde: eine ganze Zahl von 0 bis 999

Setzt die Millisekunden des angegebenen Date-Objektes nach lokaler Zeit.

- Date.setMinutes

```
var1=name.setMinutes(minutes);
```

minute: eine ganze Zahl von 0 bis 59

Setzt die Minuten des angegebenen Date-Objektes nach lokaler Zeit.

- Date.setMonth

```
var1=name.setMonth(monat  , datum));
```

monat: eine ganze Zahl von 0 (Januar) bis 11 (Dezember)

Setzt den Monat des angegebenen Date-Objektes nach lokaler Zeit.

- Date.setSeconds

```
var1=name.setSeconds(sekunde);
```

sekunde: eine ganze Zahl von 0 bis 59

Setzt die Sekunden für das angegebene Date-Objekt nach lokaler Zeit.

- Date.setTime

```
var1=name.setTime(milisekunde);
```

millisekunde: eine ganze Zahl von 0 bis 999

- Date.setUTCDate

```
var1=name.setUTCDate(datum);
```

datum: eine ganze Zahl von 1 bis 31

Setzt das Datum für das angegebene Date-Objekt nach Weltzeit. Durch Aufrufen dieser Methode werden die anderen Felder des angegebenen Datums nicht verändert. Die `getUTCDay`- und `getDay`-Methoden können jedoch einen neuen Wert zurückgeben, wenn der Wochentag aufgrund des Aufrufs dieser Methode geändert wird.

- Date.setUTCFullYear

```
var1=name.setUTCFullYear(jahr , monat ,
datum);
```

jahr: das als vierstellige Zahl angegebene Jahr, z.B. 2001

monat: eine ganze Zahl von 0 (Januar) bis 11 (Dezember) (optional)

datum: eine ganze Zahl von 1 bis 31 (optional)

Setzt das Jahr des angegebenen Date-Objektes nach Weltzeit.

Optional kann diese Methode auch Monat und Datum setzen, die durch das angegebene Date-Objekt dargestellt werden. Dabei werden keine weiteren Felder des Date-Objektes geändert. Durch Aufrufen von `setUTCFullyear` kann `getUTCDay` und `getDay` einen neuen Wert zurückgeben, wenn sich aufgrund dieser Operation der Wochentag ändert.

- Date.setUTCHours

```
var1=name.setUTCHours(stunde, minute,
sekunde, millisekunde);
```

stunde: eine ganze Zahl von 0 bis 23

minute: eine ganze Zahl von 0 bis 59 (optional)

sekunde: eine ganze Zahl von 0 bis 59 (optional)

millisekunde: eine ganze Zahl von 0 bis 999 (optional)

Setzt die Stunde für das angegebene Date-Objekt nach Weltzeit.

- Date.setUTCMilliseconds

```
var1=name.setUTCMilliseconds(millisekunde);
```

millisekunde: eine ganze Zahl von 0 bis 999

Setzt die Millisekunden für das angegebene Date-Objekt nach Weltzeit.

- Date.setUTCMinutes

```
var1=name.setUTCMinutes(minute, sekunde,
millisekunde);
```

minute: eine ganze Zahl von 0 bis 59

sekunde: eine ganze Zahl von 0 bis 59 (optional)

millisekunde: eine ganze Zahl von 0 bis 999 (optional)

Setzt die Minuten für das angegebene Date-Objekt nach Weltzeit.

- Date.setUTCMonth

```
var1=name.setUTCMonth(monat, datum);
```

monat: eine ganze Zahl von 0 (Januar) bis 11 (Dezember)

datum: eine ganze Zahl von 1 bis 31 (optional)

Setzt den Monat und optional auch den Tag (das Datum) für das angegebene Date-Objekt nach Weltzeit.

- Date.setUTCSeconds

```
var1=name.setUTCSeconds(sekunde, millise-
kunde);
```

sekunde: eine ganze Zahl von 0 bis 59

millisekunde: eine ganze Zahl von 0 bis 999 (optional)

Setzt die Sekunden für das angegebene Date-Objekt nach Weltzeit.

- Date.setYear

```
var1=name.setYear(jahr);
```

jahr: eine vierstellige Zahl, z.B. 2001

Setzt das Jahr für das angegebene Date-Objekt nach lokaler Zeit.

- Date.toString

```
var1=name.toString();
```

Gibt das angegebene Date-Objekt als Zeichenfolge zurück.

- Date.UTC

```
var1=name.UTC(jahr, monat, datum, stunde,
minute, sekunde, millisekunde);
```

jahr: das als vierstellige Zahl angegebene Jahr, z.B. 2001

monat: eine ganze Zahl von 0 (Januar) bis 11 (Dezember)

stunde: eine ganze Zahl von 0 bis 23 (optional)

datum: eine ganze Zahl von 1 bis 31 (optional)

minute: eine ganze Zahl von 0 bis 59 (optional)

sekunde: eine ganze Zahl von 0 bis 59 (optional)

millisekunde: eine ganze Zahl von 0 bis 999 (optional)

Gibt die Anzahl der Millisekunden zurück, die seit Mitternacht des 1. Januar 1979 Weltzeit und der in den Argumenten angegebenen Zeit vergangen sind. Dies ist eine statische Methode, die nicht durch ein bestimmtes Date-Objekt, sondern durch einen Konstruktor des Date-Objektes aufgerufen wird. Mit der Methode können Sie ein Date-Objekt erstellen, das der Weltzeit zugrunde liegt. Der Date-Konstruktor geht hingegen von der lokalen Zeit aus.

delete

```
delete(ziel);
```

Löscht den Inhalt eines Objektes (auch Array-Eintrages) oder einer Variable.

do... while() (Flash 4)

Führt eingeschlossene Anweisungen aus, solange die Bedingung erfüllt ist.

```
i = 0;
Do {
i++;
}while( i < 10 )
```

Diese Schleife wiederholt die eingeschlossenen Befehle (hier `I++`) zehnmal.

_droptarget (Flash 4)

Beendet die Aktion `StartDrag`, die auf ein Objekt angewandt wurde.

ZB-

```
InstanzName._droptarget;
```

Wurde die entsprechende Instanz bisher mit dem Mauscursor mitgeführt, wird diese Aktion dadurch beendet.

DuplicateMovieClip

Duplizieren von Movies
→ *68 Timeslide (Texteffekt).*

```
DuplicateMovieClip(zu kopierende Instanz,
neuer Instanzname, Tiefe);
```

Mit dem `DuplicateMovieClip`-Befehl ist es möglich, Instanzen zu kopieren. Da man ein Objekt meistens öfter kopieren möchte, wird dieser Befehl häufig in einer Schleife verwendet.

→ *removeMovieClip*

Spielprogrammierung
→ *6.1 Memory*
→ *Movieclip.duplicateMovieclip*

E

Else { (Flash 4)

Wird nur in Verbindung mit If-Bedingungen verwendet. Wird die If-Bedingung nicht erfüllt, werden die Aktionen nach `Else {` ausgeführt. Es kann immer nur einen Else-Zweig pro If-Bedingung geben.

→ *if0{}*

Else if (Bedingung) { (Flash 4)

Wird nur in Verbindung mit If-Bedingungen verwendet. Wird die If-Bedingung nicht erfüllt, wird die `Else If (Bedingung) {` überprüft.

eq (Flash 4)

→ *if0{}*

Vergleichsoperator für Strings. Liefert true oder false.

```
String1 = „Hallo";
String2 = „Welt";
String3 = „Hallo";
String1 eq String2 würde den Wert false
ergeben.
String2 eq String2 würde den Wert true erge-
ben.
```

Escape() (Flash 5)

Kodiert Zeichenketten in URL-Format. Wandelt alle Zeichen laut der URL-Encoding-Tabelle um.

Eval (Flash 4)

→ *82 Externe Dateien einlesen.*

Verwandelt Stringausdrücke in Variablen:

```
var1 = "var"add 2;
var2 = 6;
var3 = var1+var2;
var4= eval(var1) + var2 ;
```

Da `var1` einen String enthält, `wird` das Pluszeichen als Stringverkettung angesehen. `var` enthält deshalb den String `var26`.

Wenn man auf `var1 eval` anwendet, wird aus dem String der Ausdruck `var2`; dieser verweist auf die Zahl 6. Dadurch werden bei `var4` 6 + 6 addiert, was den Wert 12 ergibt. Deshalb enthält die Variable `var4` den Wert 12.

In Flash 5 gibt es nun auch eine Schreibweise, die es erlaubt, bei Pfadverweisen statt `eval()` eine andere Form zu wählen. Existieren in dem Film mehrere Movies (Movie 1 bis 9), so kann man diese Movies aus einer for-Schleife mit der folgenden Befehlszeile ansprechen:

```
eval("_root.movie"+number+".text") =
"text";
```

Hierbei wird vorausgesetzt, dass *number* eine laufende Zahl ist, die bis 9 hochgezählt wird.

In Flash 5 kann man dies auch so schreiben:

```
_root["movie"+number].text = "horror";
```

Evaluate (Flash 5)

Kapitel 2.4: Neuerungen in Flash ActionScript

Erlaubt die Eingabe von selbstdefinierten Funktionen im Normalmodus der ActionScripteingabe. Im Expertmodus existiert dieser Befehl nicht, da dort die Eingabe von Funktionen direkt möglich ist.

F

_focusrect (Flash 4)
Schaltet das gelbe Fokussierrechteck, das beim Drücken der Tabulatortaste erscheint und zur Navigationshilfe dient, für alle Navigationselemente aus/ein.
Navigationshilfe ein (Standard):

```
_focusrect = true;
```

Navigationshilfe aus:

```
_focusrect = false;
```

for (Flash 4)

```
for (init; Bedingung; next) {
     Anweisung;
}
```

Eine Schleife, die einen Wert fortlaufend verändert und so lange alle enthaltenen Anweisungen ausführt, bis die Bedingung erfüllt ist.

zB:

```
for (c=0; c<maxp+1; c++) {
     duplicateMovieClip (kreis, "kreis"+c,
c);
}
```

for..in (Flash 5)

```
for (element in object){
...
}
```

Geht alle Elemente des Objektes durch und führt die in geschweiften Klammern {} stehenden Anweisungen aus.

_framesloaded (Flash 4)
Mit der Eigenschaft `_framesloaded` kann man die Anzahl der bereits geladenen Frames des jeweiligen SWFs abfragen. `_frameloaded` unterteilt SWFs nicht nach Szenen.
→ *46 Preloader*
In dem folgenden Beispiel wird abgefragt, ob die ersten 40 Frames eingeladen wurden, da diese Frames das Intro enthalten. Das Intro wird abgespielt, wenn die ersten 40 Frames eingeladen sind und in Frame 41 wird abgefragt, ob der komplette Film eingeladen wurde.

- Frame 2

```
if (_framesloaded>40) {
    gotoAndPlay (1);
} else {
    gotoAndPlay (3);
}
```

- Frame 40

```
if (_framesloaded >= _totalframes) {
     gotoAndPlay (42);
} else {
     gotoAndPlay (40);
}
```

FSCommand (Flash 4)

```
fscommand( Befehl, Argument);
```

Mittels FSCommand rufen Sie externe Scripte auf. Achten Sie beim Veröffentlichen auf die Einstellung „Flash mit FS Command“.

Function (Flash 5)

```
function
Name (Argumente) {
              Befehle;
}
```

Aufruf:

```
Name(Argumente);
```


→ *6.1 Memory – Spieleranzeige.*

Mit Funktionen kann man mehrere Befehle zu einem Block zusammenfassen und diesen Block bei Bedarf aufrufen.
Flash-4-User → *Call*
→ *return();*

G

ge

```
Zeichenkette1 ge Zeichenkette2
```

Der für Flash-4-Scripts erforderliche „größer gleich"-Vergleichsoperator. ge war bei Vergleichen zwischen Zeichenfolgen erforderlich; ge liefert Boolean zurück.

getBytesLoaded

→ *MovieClip.getBytesLoaded*

getBytesTotal

→ *MovieClip.getBytesTotal*

GetProperty() (Flash 4)

```
GetProperty( Instanzname abzufragende
Eigenschaft)
```

Mit `getProperty` kann man die Eigenschaften von Filmsequenzen abfragen: Beispielsweise werden die X- und Y-Positionen des Movieclips abgefragt.

```
var1=getProperty ("instanzname", _x);
var2=getProperty ("instanzname", _y);
```

`var1` entspricht der X-Position des Movieclips *Instanzname,* `var2` seiner Y-Position. In Flash 5 sollte man die Punktsyntax benutzen.

```
var1=instanzname._x;
var2=instanzname._y;
```

getTimer() (Flash 4)

Mit der Funktion `getTimer()` kann man den Zeitraum seit Start des Films in Millisekunden abfragen.

→ *6.1 Memory –Spieldaueranzeige*

Man kann die Funktion benutzen, um verschiedene Timingfunktionen zu ermöglichen.

GetURL() (Flash 4)

```
getURL ("URL", "Framename", "POST");
```

→ *92 Timeout (Screensaver-Effekt).*

Der Framename sowie das Senden von Variablen mit `Post` oder `Get` sind optional.

Die URL-Angabe kann relativ oder absolut erfolgen. Relative Angaben beziehen sich immer auf den Pfad, von dem aus die HTML-Datei aufgerufen wurde. Angenommen, Sie haben auf Ihrem Server folgende Verzeichnisstruktur angelegt:

Ihre zuerst gestartete HTML-Datei liegt in dem Verzeichnis *Startseite,* die Datei haupt.html aus dem Verzeichnis *Hauptseite* rufen Sie mit einer relativen Pfadangabe auf:

```
getURL ("../Hauptseite/haupt.html");
```

Die Punkte werden benutzt, um das Verzeichnis *Startseite* zu verlassen.

Eine absolute URL gibt immer den kompletten Pfad an ZB:

```
getURL ("http://www.Server.de/HTML/Haupt-
seite/haupt.html");
```

Wenn Sie eine Seite mit Framesets erstellt haben, können Sie den Frames des Framesets unter HTML Namen zuweisen. Unter Flash könne Sie die Framesets über diese Namen ansprechen.

```
getURL ("http://www.Server.de/HTML/Haupt-
seite/haupt.html", "Framesetname");
```

Ihr Frameset unter HTML sollte ungefähr so aussehen :

```
<html>
<head>
<title></title>
<meta http-equiv="Content-Type" con-
tent="text/html; charset=iso-8859-1">
</head>
<frameset rows="20%, *" border="0" frame-
spacing="0" frameborder="NO">
  <frame src="oben.html" name="Flash" mar-
ginwidth="0" marginheight="0" scrol-
ling="NO" noresize>
<frameset cols="30%, *" border="0" frame-
spacing="0" frameborder="NO">
  <frame src="auswahl.html" name="Leer1"
marginwidth="0" marginheight="0" scrol-
ling="NO" noresize>
  <frame src="haupt.html" name="Leer" mar-
ginwidth="0" marginheight="0" scrol-
```

```
ling="NO" noresize>
</frameset>
</frameset>
<noframes><body bgcolor="#FFFFFF">
Fuer diese Seite benoetigen Sie einen Brow-
ser der Framsets unterstuetzt.
</body></noframes>
</html>
```

Mit `getURL()` kann auch Variablen an andere Adressen schicken. Dies geht mit `Get` oder `Post`.

```
getURL ("http://www.Server.de/Hauptseite/
haupt.html", "", "GET");
```

Wenn man die Variablen mit `Get` verschickt, werden diese direkt mit an die Adresse gehängt. Der Adressenaufruf würde dann so aussehen:

```
http://Server.de?Vari-
able=wert&Variable2=wert2
```

Wenn man die Variablen hingegen mit *Post* verschickt, werden diese separat übermittelt.

```
getURL ("http://www.Server.de/Hauptseite/
haupt.html", "", "POST");
```

→ *MovieClip.getURL*

GetVersion() (Flash 5)

Diese Funktion überprüft, welches Betriebssystem und welche Version des Flashplayers auf dem Rechner des Users installiert ist. Die Ausgabe der Funktion könnte z.B. folgendermaßen aussehen: WIN 5,0,30,0

→ *112 Zugriffszähler (Counter)*

GetVersion → *65 Flash-Detection.*

WIN für Windows
5.0 für die 5 Version des Flashplayers
30.0 für die genauere Bestimmung der Version
5.0.30.0

globalToLocal → *MovieClip.globalToLocal*

GotoAndPlay (Flash 4)

```
gotoAndPlay ("Szene 1", Framenummer);
```

Flash beginnt, von dem angegebenen Frame aus abzuspielen bzw. springt zu diesem Frame.

→ *MovieClip.GotoAndPlay*

GotoAndStop (Flash 4)

```
gotoAndStop ("Szene 1", Framenummer);
```

Flash springt zu diesem Frame und stoppt.

→ *MovieClip.GotoAndStop*

gt

```
Zeichenkette1 gt Zeichenkette2
```

Der für Flash-4-Scripts erforderliche „größer als"-Vergleichsoperator. gt war bei Vergleichen zwischen Zeichenfolgen erforderlich; gt liefert Boolean zurück.

H

hide

→ *Mouse.hide*

_highquality

```
_highquality = Wert;
```

Verändert die Wiedergabequalität des Flashfilms. Gültige Werte sind:

- 0: Anti-Aliasing im 2x2-Raster, keine geglätteten Bitmaps
- 1: Anti-Aliasing im 4x4-Raster, in statischen Filmen geglättete Bitmaps
- 2: Anti-Aliasing im 4x4-Raster, grundsätzlich geglättete Bitmaps

_height

```
Instanzname._height
```

Liefert als Wert die Höhe der Instanz gemessen in Pixeln zurück, bzw. kann neu gesetzt werden. Die Proportionen werden nicht berücksichtigt.

Abfrage von Eigenschaften
→ *6.2 Bewegung.*

hstTest

→ *MovieClip.hstTest*

I

if

```
if (Bedingung){
...}
}
```

Führt Befehle innerhalb der Klammer einmal aus, sofern die Bedingung den Wert *true* (Boolean) hat.

IfFrameLoaded

```
ifFrameLoaded(Szene,FrameBez){
...
}
```

Führt Befehle innerhalb der Klammer einmal aus, sofern der Frame FrameBez (angegeben werden kann die Frame-Nummer oder auch die Bezeichnung) in der Szene geladen wurde.

ifFramesLoaded
→ *46 Preloader*
– Einfacher Preloader.

#include

```
#include "Actionscript.as";
```

Schließt zusätzliches ActionScript mit in den Film ein.

Infinity

```
Infinity
```

Größte mögliche Zahl nach ECMA-262 Standard.

Int

```
int(Wert);
```

Liefert als Wert den ganzzahligen Anteil des Wertes zurück.

Auf- und Abrunden
→ *39 Mailfunktionen.*

isFinite

```
isFinite(Ausdruck);
```

Wertet den Ausdruck aus und überprüft, ob dieser laut IEEE-754-Standard endlich ist; Rückgabewerte sind hier Boolean (true oder false).

isNaN

```
isNaN(Ausdruck);
```

Überprüft bei einem Ausdruck, inwieweit es sich bei diesem um keine Zahl handelt. Rückgabewerte sind hier Boolean (true oder false).

join() →*Array.join()*

K

Key (Objekt)

Keyobjekt
→ *6.4 Einbinden von zusätzlichen Movies – Tastaturabfrage.*

- Key.BACKSPACE
  `Key.BACKSPACE`

Eigenschaft, enthält den Wert der Rückschritttaste ⟵

- Key.CAPSLOCK
  `Key.CAPSLOCK`

Eigenschaft, enthält den Wert der Feststelltaste ⇪.

- Key.CONTROL
  `Key.CONTROL`

Eigenschaft, enthält den Wert der Taste Steuerung Strg.

- Key.DELETEKEY
  `Key.DELETEKEY`

Eigenschaft, enthält den Wert der Taste Entfernen Entf.

- Key.DOWN
  `Key.DOWN`

Eigenschaft, enthält den Wert der Taste Pfeil-nach-unten ↓.

- Key.END
  `Key.END`

Eigenschaft, enthält den Wert der Taste Ende Ende.

- Key.ENTER
  `Key.ENTER`

Eigenschaft, enthält den Wert der Taste Eingabe ↵.

- Key.ESCAPE
  `Key.ESCAPE`

Eigenschaft, enthält den Wert der Taste Escape [Esc].

- Key.getAscii
  `Key.getAscii();`

Liefert den ASCII-Code der (zuletzt) gedrückten Taste zurück.

- Key.getCode
  `Key.getCode()`

Liefert den Tastaturcode-Wert der (zuletzt) gedrückten Taste zurück.

- Key.HOME
  `Key.HOME`

Eigenschaft, enthält den Wert der Taste Position Eins [Pos1].

- Key.INSERT
  `Key.INSERT`

Eigenschaft, enthält den Wert der Taste Einfügen [Einfg].

- Key.isDown
  `Key.isDown(Tastencode);`

Liefert *true* zurück wenn die dem Tastencode entsprechende Taste gedrückt wurde, *false* wenn nicht.

- Key.isToggled
  `Key.isToggled`

Eigenschaft, enthält den Status der Feststelltaste. Liefert Boolean zurück (*true*=aktiviert, *false*=deaktiviert).

- Key.LEFT
  `Key.LEFT`

Eigenschaft, enthält den Wert der Taste Pfeil-nach-links [←].

- Key.PGDN
  `Key.PGDN`

Eigenschaft, enthält den Wert der Taste Bild-nach-unten [Bild ↓].

- Key.PGUP
  `Key.PGUP`

Eigenschaft, enthält den Wert der Taste Bild-nach-oben [Bild ↑].

- Key.RIGHT
  `Key.RIGHT`

Eigenschaft, enthält den Wert der Taste Pfeil-nach-rechts [→].

- Key.SHIFT
  `Key.SHIFT`

Eigenschaft, enthält den Wert der Umschalttaste [⇧].

- Key.SPACE
  `Key.SPACE`

Eigenschaft, enthält den Wert der Leertaste [Leertaste].

- Key.TAB
  `Key.TAB`

Eigenschaft, enthält den Wert der Tabulatortaste [⇆].

- Key.UP
  `Key.UP`

Eigenschaft, enthält den Wert der Taste Pfeil-nach-oben [↑].

L

le

`Zeichenkette1 le Zeichenkette2`

Der für Flash-4-Scripte erforderliche „kleiner gleich"-Vergleichsoperator. le war bei Vergleichen zwischen Zeichenfolgen erforderlich; le liefert Boolean zurück.

Length()

→ *Array.length*
→ *String.length()*

Length()-Abfrage (Flash 4),
→ *6.1 Memory*
– Anzeige der Uhrzeit.

_level (Flash 4) → *_root*

loadMovie

→ *MovieClip.loadMovie*

loadVariables

→ *MovieClip.loadVariables*

LocalToGlobals

→ *MovieClip.LocalToGlobal*

lt

```
Zeichenkette1 lt Zeichenkette2
```

Der für Flash-4-Scripte erforderliche „kleiner als"-Vergleichsoperator. lt war bei Vergleichen zwischen Zeichenfolgen erforderlich; lt liefert Boolean zurück.

M

Math.abs (Flash 4)

```
Ergebnis = Math.abs(Wert);
```

Gehört zu dem Math-Objekt, liefert den positiven Wert des Argumentes zurück.

ZB:

```
X = -10;
Y = Math.abs(X);
```

Y wurde nun der Wert 10 zugeordnet.

Math.acos

```
Ergebnis = Math.acos(Wert);
```

Gehört zu dem Math-Objekt, liefert als Wert den Arkuskosinus des Argumentes zurück. Werte werden im Bogenmaß angegeben. Der Umrechnungsfaktor zu Grad ist π/180.

Math.asin

```
Ergebnis = Math.acos(Wert);
```

Gehört zu dem Math-Objekt, liefert als Wert den Arkussinus des Argumentes zurück. Werte werden im Bogenmaß angegeben. Der Umrechnungsfaktor zu Grad ist π/180.

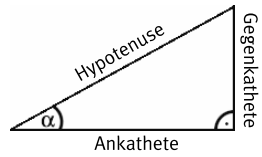
Math.atan

```
Ergebnis = Math.acos(Wert);
```

Gehört zu dem Math-Objekt, liefert als Wert den Arkustangens des Argumentes zurück. Werte werden im Bogenmaß angegeben. Der Umrechnungsfaktor zu Grad ist π/180.

Math.atan2

```
Ergebnis = Math.atan2(WertY, WertX);
```



Gehört zu dem Math-Objekt, liefert als Wert den Arkustangens des durch die beiden Argumente beschriebenen rechtwinkligen Dreiecks zurück. Für den Innenwinkel, der hier verwendet wird, ist `WertY` Ankathete und `WertX` Gegenkathete. Werte werden im Bogenmaß angegeben. Der Umrechnungsfaktor zu Grad ist π/180.

Math.ceil

```
Ergebnis = Math.ceil(Wert);
```

Gehört zu dem Math-Objekt, liefert als Wert die Obergrenze des Argumentes zurück. Das Ergebnis ist die dem Wert nächsthöhere ganze Zahl. Eine Ausnahme besteht, wenn der Wert schon ganzzahlig ist.

Math.cos

```
Ergebnis = Math.cos(Wert);
```

Gehört zu dem Math-Objekt, liefert als Wert den Kosinus des Argumentes zurück. Werte werden im Bogenmaß angegeben. Der Umrechnungsfaktor zu Grad ist π180.

Math.E

```
Ergebnis = Math.E;
```

Mathematische Konstante und Basis zu `Math.log(x)`, (natürlicher Logarithmus).

Math.exp

```
Ergebnis = Math.exp(Wert);
```

Gehört zu dem Math-Objekt, liefert als Wert *eWert. Setzt man für Wert = 1, kann man dadurch auch e erzeugen.*

Math.floor

```
Ergebnis = Math.floor(Wert);
```

Gehört zu dem Math-Objekt, liefert als Wert den ganzzahligen Anteil des Arguments zurück.

Math.log

```
Ergebnis = Math.log(Wert);
```

Gehört zu dem Math-Objekt, liefert als Wert den natürlichen Logarithmus des Argumentes zurück, also den Logarithmus zur Basis *e*.

Math.LOG2E

```
Ergebnis = Math.LOG2E;
```

Gehört zu dem Math-Objekt, liefert als Wert den natürlichen Logarithmus von 2 zurück.

→ *Math.LN2;*

Math.LOG10E

```
Ergebnis = Math.LOG10E (Wert);
```

Gehört zu dem Math-Objekt, liefert als Wert den natürlichen Logarithmus von 10 zurück.

→ *Math.LN10;*

Math.LN2

```
Ergebnis = Math. LN2;
```

Gehört zu dem Math-Objekt, liefert als Wert den natürlichen Logarithmus von 2 zurück.

Math.LN10

```
Ergebnis = Math.LN10;
```

Gehört zu dem Math-Objekt, liefert als Wert den natürlichen Logarithmus von 10 zurück.

Math.max

```
Ergebnis = Math.max(Wert1, Wert2);
```

Gehört zu dem Math-Objekt, liefert als Wert das größere Argument zurück.

Math.min

```
Ergebnis = Math.min(Wert1, Wert2);
```

Gehört zu dem Math-Objekt, liefert als Wert das kleinere Argument zurück.

Math.PI

```
Ergebnis = Math.PI;
```

Gehört zu dem Math-Objekt, liefert als Wert die Konstante $\pi = 3{,}14\ldots$ zurück.

Math.pow

```
Ergebnis = Math.pow(Wert1, Wert2);
```

Gehört zu dem Math-Objekt, liefert $Wert1^{Wert2}$ zurück.

Math.random

```
Ergebnis = Math.random();
```

Gehört zu dem Math-Objekt, liefert als Wert eine Zufallszahl zwischen 0,0 und 1,0 zurück.

Math.round

```
Ergebnis = Math.round(Wert);
```

Gehört zu dem Math-Objekt, liefert als Wert das auf- oder abgerundete Argument zurück.

Math.sin

```
Ergebnis = Math.sin(Wert);
```

Gehört zu dem Math-Objekt, liefert als Wert den Sinus des Argumentes zurück. Werte werden im Bogenmaß angegeben. Der Umrechnungsfaktor zu Grad ist $\pi/180$.

Math.sqrt

```
Ergebnis = Math.sqrt (Wert);
```

Gehört zu dem Math-Objekt, liefert als Wert die Quadratwurzel des Argumentes zurück. Es muss gelten, dass der Wert größer oder gleich 0 ist.

Math.SQRT1_2

```
Ergebnis = Math.SQRT1_2;
```

Gehört zu dem Math-Objekt, liefert als Konstante die Quadratwurzel von ½.

Math.SQRT2

```
Ergebnis = Math.SQRT2;
```

Gehört zu dem Math-Objekt, liefert als Konstante die Quadratwurzel von 2

Math.tan

```
Ergebnis = Math.tan (Wert);
```

Gehört zu dem Math-Objekt, liefert als Wert den Tangens des Argumentes zurück. Werte werden im Bogenmaß angegeben. Der Umrechnungsfaktor zu Grad ist $\pi/180$.
Tangens(Wert) = Sin(Wert)/Cos(Wert)

maxscroll (Flash 4)

```
Ergebnis = Textfeld.maxscroll;
```

Liefert als Ergebnis den höchstmöglichen Scrollwert des Textfeldes. Der Wert ist schreibgeschützt.

mbchr (Flash 4)

```
mbchr(Wert);
```

Verwandelt eine Zahl (Wert) in ein Multibyte-Zeichen (-character).

→ *String.fromCharCode.*

mblength (Flash 4)

```
mblength(Wert);
```

Gibt als Ergebnis die Zeichenlänge des Wertes zurück.

mbord (Flash 4)

```
mbord(Wert);
```

Verwandelt ein Multibyte-Zeichen (-character) in eine Multibyte-Zahl (Wert).

→ *String.fromCharCode.*

mbsubstring (Flash 4)

```
mbsubstring(Wert, Start, Laenge);
```

Gibt als Ergebnis den Auszug des `Wertes` zurück, beginnend von `Start` `Laenge` viele Zeichen.

Mouse.hide

```
Mouse.hide();
```

Blendet den Mauszeiger über dem Film aus.

Mousehide

→ *70 Mauseffekte/Maustrailer*
– *Mousehide.*

Mouse.show

```
Mouse.show();
```

Blendet den Mauszeiger über dem Film ein.

MovieClip.attachMovie

```
Filmsequenz.attachMovie( idName, neue-
Instanz, tiefe);
```

Hängt einen Movieclip namens `idName` aus einer Library, legt diesen dann unter dem Namen `neue-Instanz` in eine vorgegebene `tiefe`.

AttachMovie(),

→ *6.4 Memory*
– *Spieleranzeige.*

MovieClip.duplicateMovieClip (Flash 4)

Filmsequenz.duplicateMovieClip(neueInstanz, tiefe);

Dupliziert die Filmsequenz und legt den neuen Film unter dem Instanznamen `neueInstanz` ab. Werden verschiedene Instanzen in die selbe `tiefe` geladen, wird die alte Instanz gelöscht.

→ *duplicateMovieClip*
Duplizieren von Movies
→ *68 Timeslide (Texteffekt).*

MovieClip.getBounds (Flash 5)

```
Filmsequenz.getBounds(Bezugssystem);
```

Gibt die maximalen X-, Y-Abmessungen der Filmsequenz wieder. Das Bezugssystem meint hier einen Pfad, auf dessen Nullpunkt Bezug genommen werden soll.

MovieClip.getBytesLoaded (Flash 5)

```
Filmsequenz.getBytesLoaded();
```

Gibt die Anzahl der bereits geladenen Bytes der Filmsequenz zurück.

GetBytesLoaded

→ *46 Preloader*
– *Zusätze für Preloader.*

MovieClip.getBytesTotal (Flash 5)

```
Filmsequenz.getBytesLoaded();
```

GetBytesLoaded

→ *46 Preloader*
– *Zusätze für Preloader.*

MovieClip.getURL

```
Filmsequenz.getURL(URL[, Fenster, Trans-
fer]);
```

Lädt eine *URL* in ein definiertes *Fenster*, das sie in einem Frameset per `<frame src="../.." name="Ziel-Fenster">` angeben. Die Variablen werden der *URL* mit z.B. `test.asp?Wert=1&Wert1=2` übergeben, doch die Art der Übertragung wird in *Transfer* mit „`POST`“ oder „`GET`“ festgelegt.

→ *getURL*

MovieClip.globalToLocal

```
Filmsequenz.globalToLocal;
```

Ordnet den Koordinatenursprung der Hauptbühne einer Filmsequenz zu.

MovieClip.gotoAndPlay

```
Filmsequenz.gotoAndPlay(FrameNr);
```

Startet das Abspielen einer Filmsequenz im Frame `FrameNr`.

→ *gotoAndPlay*

MovieClip.gotoAndStop

```
Filmsequenz.gotoAndStop(FrameNr);
```

Geht zum Frame *FrameNr* einer Filmsequenz und stoppt.

→ *gotoAndStop*

MovieClip.hitTest

```
Instanzname.hitTest(Ziel) ;
```

Überprüft, ob sich zwei Instanzen überschneiden. Die Rahmen und nicht die Form des eigentlichen Instanzinhaltes sind hier entscheidend.

ZB:

```
if (Instanz1, hittest(Instanz2)){
...
}
```

HitTest,
→ *6.4 Einbinden von zusätzlichen Movies (Objekten) – Kollisionsdetection.*

Sobald sich Instanz1 und Instanz2 überschneiden, wird die Bedingung erfüllt, die Befehle innerhalb der geschweiften Klammern {} ausgeführt.

MovieClip.loadMovie

```
Filmsequenz.loadMovie.(URL[, Transfer]);
```

Lädt eine .swf-Datei nachträglich in den laufenden Film ein. *Transfer* gibt auch hier die Art der möglichen Variablenversendung an „`GET`“ oder „`POST`“.

LoadMovie
→ *46 Preloader – Laden einzelner SWFs.*

MovieClip.loadVariables

```
Filmsequenz.loadVariables.(URL[, Transfer]);
```

Lädt die Variablen einer externen Datei nachträglich in den laufenden Film ein. `Transfer` gibt auch hier die Art der möglichen Variablenversendung an („`GET`“ oder „`POST`“).

LoadVariables,
→ *5.5 CGI und folgende.*

MovieClip.localToGlobal

```
Filmsequenz.localToGlobal(x_y_pos);
```

Ordnet einen beliebigen Koordinatenursprung der Hauptbühne zu. Die Koordinaten werden in einem Objekt übergeben mit den Werten x und y. Das fertige ActionScript sieht wie folgt aus:

```
x_y_pos = new Object;
x_y_pos.x=100;
x_y_pos.y=100;
localToGlobal(x_y_pos) ;
```

Der Koordinatenursprung wurde nun 100 Pixel nach rechts und 100 Pixel nach unten verschoben, relativ zum Bezugsnullpunkt des Koordinatensystems, aus dem die Funktion aufgerufen wird.

MovieClip.nextFrame

```
Filmsequenz.nextFrame();
```

Lässt den Film aus dem aktuellen Frame in den nächsten springen.

MovieClip.play

```
Filmsequenz.play();
```

Gibt eine Filmsequenz wieder.

MovieClip.prevFrame

```
Filmsequenz.prevFrame();
```

Lässt den Film aus dem aktuellen Frame in den vorherigen springen.

MovieClip.removeMovieClip

```
Filmsequenz.removeMovieClip();
```

Entfernt die Instanz einer Filmsequenz.

MovieClip.startDrag

```
Filmsequenz.startDrag(center, links,
rechts, oben, unten);
```

StartDrag,
➜ *6.2 Bewegung – Einfluss durch den User oder auch ➜ 74 Soundeffekte.*

Heftet die Filmsequenz an die Filmsequenz.

`center` = true: Die Filmsequenz wird in ihrem Mittelpunkt an die Mauskoordinaten geheftet – optional.

`center` = false: Die Filmsequenz wird relativ zu den letzten Mauskoordinaten an diese geheftet – optional.

`links, rechts, oben, unten`: relative Verschiebung, optional.

➜ *startdrag*

MovieClip.stop

```
Filmsequenz.stop();
```

Stoppt die Filmsequenz.

MovieClip.stopDrag

```
Filmsequenz.stop();
```

Stoppt die mittels `startDrag` gestartete Ziehoperation.

MovieClip.swapDepths

```
Filmsequenz.swapDepths(tiefe);
Filmsequenz.swapDepths(ziel);
```

Legt eine Filmsequenz in einer z-Ebene ab (`tiefe`) oder tauscht mit einer vorhandenen, falls diese z-Ebene schon belegt ist.

MovieClip.unloadMovie

```
Filmsequenz.unloadMovie();
```

Entlädt eine Filmsequenz.

N

_name

```
Instanzname._name;
```

Liefert den Wert `Name` als Eigenschaft einer Instanz zurück.

NaN

Not a Number, keine Zahl. Wird ausgegeben, wenn aus einer Operation nicht wie erwartet eine Zahl hervorgeht.

ne

Not Equal, nicht gleich. Überprüft zwei Zeichenketten daraufhin, ob diese identisch sind oder nicht.

`(ZeichenEins ne ZeichenZwei)` liefert Boolean zurück, hier true (wahr).

new

```
new konstruktor();
```

Legt einen neuen Objekttyp an.

ZB:

```
function Familie(Frau, Sohn){
this.Frau=Frau;
this.Sohn=Sohn;
}
Michael=new Familie(Martha, Markus);
Markus=new Familie(Maria, Matthias);
```

Hier werden objektorientiert Strukturen erstellt.

newline

```
newline;
```

Fügt einen Zeilenumbruch in das ActionScript ein.

nextFrame

```
nextFrame();
```

Lässt den Film aus dem aktuellen Frame in den nächsten springen.

nextScene

```
nextScene();
```

Lässt den Film aus der aktuellen Szene in die nächste springen.

not

```
not Bedingung
```

Logisches „nicht“, kann auch durch „! Bedingung“ erzeugt werden. Kehrt den Wahrheitsgehalt der Aussage um.

null

```
null
```

Mit diesem Ausdruck kann man die Existenz von Objekten und Werten überprüfen. Sind diese `null`, sind sie im Film noch nicht existent.

Number

```
var1 = "5";
var2 = "6";
var3 = var1+var2;
var4= number(var1)+number(var2);
var entspricht 56.
var 4 entspricht 11.
```

wandelt einen String in eine Zahl um

Number.MAX_VALUE

```
Number.MAX_VALUE
```

Größtmögliche Zahl nach IEEE-754 Standard, 1,79E+308.

Number.MIN_VALUE

```
Number.MIN_VALUE
```

Kleinstmögliche Zahl nach IEEE-754 Standard, 5E-324.

Number.NaN

```
Number.NaN
```

Eigenschaft, → *NaN.*

Number.NEGATIVE_INFINITY

```
Number.NEGATIVE_INFINITY
```

Wird ausgegeben wenn der zulässige Zahlenraum nach IEEE-754-Standard negativ überschritten wird.

Number.POSITIVE_INFINITY

```
Number.POSITIVE_INFINITY
```

Wird ausgegeben, wenn der zulässige Zahlenraum nach IEEE-754-Standard positiv überschritten wird.

Number.toString

```
Zahl.toString(Basis);
```

Verändert die Grundzahl eines Wertes.

```
nummer = new Number (1978);
(1978).toString(2);
```

Gibt eine Zeichenkette mit der binären Darstellung des Wertes 1978 zurück.

Number.valueOf

```
Zahl.valueOf();
```

Gibt den Grundwerttyp der Zahl zurück.

O

Object

```
new Object();
```

Konstruktor, erstellt ein neues Objekt.

Object.toString

```
Objekt.toString();
```

Gibt ein Objekt als Zeichenfolge zurück.

Object.valueOf

```
Objekt.valueOf();
```

Gibt den Grundwerttyp eines Objektes zurück.

OnClipEvent

```
onClipEvent(event);{
...
}
```

→ *2.4 Neuerungen in ActionScript*
→ *1.3 Programmiergrundlagen*

Frägt Ereignisse der Filmsequenz ab und führt ggf. angegebene Aktionen aus.

on(mouseEvent)

```
on(mouseEvent);{
...
}
```

Fragt Ereignisse der Maus ab und führt ggf. angegebene Aktionen aus.

or

```
Bedingung1 or Bedingung2
```

Logisches „oder“, liefert Werte nach folgendem Schema zurück:

Wert1 || Wert2 oder Wert1 or Wert2

Wert 1	Wert 2	Ergebnis
true	true	true
true	false	true
false	true	true
false	false	false

ord

```
ord(Zeichen);
```

Gibt den ASCII-Wert eines Zeichens zurück.

P

_parent

```
_parent.eigenschaft = Wert
```

Liefert in der Hierarchie Werte des eigenen bzw. übergeordneten Objektes zurück.

parseFloat

```
parseFloat(Zeichenfolge);
```

Verwandelt die Zeichenfolge in eine Gleitkommazahl.

parseInt

```
parseInt(ausdruck, basis);
```

Verwandelt den Ausdruck (`ausdruck`) in eine ganze Zahl zur Basis (`basis`).

play

```
play();
```

Spielt den Film weiter ab.

pop

→ *Array.pop*

prevFrame

```
prevFrame();
```

Lässt den Film aus dem aktuellen Frame in den vorherigen springen und stoppen.

prevScene

```
prevScene();
```

Lässt den Film aus der aktuellen Szene in die vorherige springen.

print

```
print (ziel, "bmovie");
print (ziel, "bmax");
print (ziel, "bframe");
```

Druck eine Instanz (`ziel`). Bestimmte Frames, die gedruckt werden sollen, können durch die Bezeichnung #p ausgewählt werden. Die Daten werden, sofern möglich, in Vektoren an den Drucker gesendet.

`bmovie`: Legt die Begrenzungen eines einzelnen Bildes als Rand für den Druckbereich aller Bilder fest.

`bmax`: Vergrößert den druckbaren Bereich auf ein Maximum.

`bframe`: Legt die Begrenzungen jedes einzelnen Bildes als Ränder für den Druckbereich fest.

printAsBitmap

```
printAsBitmap (ziel, "bmovie");
printAsBitmap (ziel, "bmax");
printAsBitmap (ziel, "bframe");
```

Druck eine Instanz (`ziel`). Bestimmte Frames, die gedruckt werden sollen, können durch die Bezeichnung #p ausgewählt werden. Die Daten werden als Bild an den Drucker gesendet. Für die Optionen → *„print“.*

push

→ *Array.push*

Q

_quality

```
_quality = Wert;
```

Verändert die Wiedergabequalität des Flashfilms.

LOW: kein Anti-Aliasing, keine geglätteten Bitmaps

MEDIUM: Anti-Aliasing im 2x2-Raster, keine geglätteten Bitmaps

HIGH: Anti-Aliasing im 4x4-Raster, in statischen Filmen geglättete Bitmaps

BEST: Anti-Aliasing im 4x4-Raster, grundsätzlich geglättete Bitmaps

R

random

```
random();
```

Wird kein Wert eingegeben, so wird ein zufälliger Wert zwischen 0 und 1 zurückgeliefert; gibt man hingegen einen bestimmten Wert ein (`random(Wert)`), so wird ein Wert zwischen 0 und diesem Wert zurückgeben.

removeMovieClip

```
removeMovieClip(Instanzname);
```

Entfernt die Instanz (Instanzname) aus dem Filmsequenz.

return()

```
return;
```

Return()
→ *6.1 Memory – Spieleranzeige.*

Kehrt zurück zur übergeordneten Funktion, kann ggf. auch Werte zurückgeben. Eine Funktion, die den Aufruf `return 3;` beinhaltet, hat den Wert 3.

_root

```
_root;
```

Entspricht dem Ursprung der Pfade, bekannt aus Flash 4 als `_level0`.

_rotation

```
Instanzname._rotation = Wert;
```

Legt den Wert der Drehung einer Instanz fest.

_reverse

→ *Array.reverse*

S

scroll

```
Variable.srcoll = Wert;
```

Legt in Bezug auf ein Textfeld mit Wert die oberste sichtbare Zeile fest.

Selection.getBeginIndex

```
Selection.getBeginIndex();
```

Gibt den Anfang des Auswahlbereiches (Rückgabewerte 0, 1, ..., n) zurück; wenn keine Auswahl getroffen wurde, wird -1 zurückgegeben.

Selection.getCaretIndex

```
Selection.getCaretIndex();
```

Gibt die Position des Cursors im Auswahlbereich (Rückgabewerte 0, 1, ..., n) zurück; wenn dort kein Cursor vorhanden ist, wird -1 zurückgegeben.

Selection.getEndIndex

```
Selection.getEndIndex();
```

Gibt das Ende des Auswahlbereiches (Rückgabewerte 0, 1, ..., n) zurück; wenn keine Auswahl getroffen wurde, wird -1 zurückgegeben.

Selection.getFocus

```
Selection.getFocus();
```

Gibt den Namen des ausgewählten Feldes zurück; wenn keins gewählt ist, wird –1 zurückgegeben.

Selection.setFocus

```
Selection.setFocus("Variable");
```

Gibt den Namen des zu aktivierenden Feldes an.

Selection.setSelection

```
Selection. setSelection (start, ende);
```

Gibt den Auswahlbereich eines Textfeldes an. Für `start` und `ende` sind Werte von 0 bis n zulässig, wobei `start < ende` gilt.

set

```
set(Variable, Ausdruck)
```

Ordnet einem Ausdruck eine Variable zu, gleichbedeutend mit Ausdruck = Variable.

SetProperty() (Flash 4)

```
GetProperty( Instanzname, zu setzende
Eigenschaft, Wert)
```

Mit `getProperty` kann man auf die Eigenschaften von Filmsequenzen zugreifen – z.B. die X- und Y-Koordi-

naten verändern. Hier werden die X- und Y-Positions des Movieclips auf 100 gesetzt.

```
setProperty ("instanzname", _x, "100");
setProperty ("instanzname", _y, "100");
```

show

→ *Mouse.show*

shift

→ *Array.shift*

slice

→ *Array.slice*

sort

→ *Array.sort*

Sound

```
Soundobj = new Sound();
```

Sound

→ *62 Sound und → 74 Soundeffekte.*

Sound.attachSound

```
Soundobj.attachSound("idName");
```

Fügt dem Soundobjekt einen neuen Sound hinzu.

Sound.getPan

```
Soundobj.getPan();
```

Gibt die Balanceeinstellungen zurück (-100 = links ... 100 = rechts).

Sound.getTransform

```
Soundobj.getTransform();
```

Gibt die mittels `Sound.setTransform` gesetzten Einstellungen zurück.

Sound.getVolume

```
Soundobj.getVolume();
```

Gibt die mittels `Sound.setVolume` gesetzte Lautstärke zurück.

Sound.setPan

```
Soundobj.setPan(bal);
```

Setzt die Balance eines Soundobjekts. Zulässige Werte liegen zwischen -100 (linker Kanal) und 100 (rechter Kanal).

Sound.setTransform

```
Sound.setTransform(Transformobj);
```

Gibt mittels eines Objektes eine Soundtransformation an. In der Praxis sieht das wie folgt aus:

```
soundTransform = new Object;
soundTransform.ll = 100;
soundTransform.lr = 100;
soundTransform.rr = 0;
soundTransform.rl = 0;
Sound.setTransform(soundTransform);
```

Die „Anhänge“ ll, lr, rr, rl geben die prozentuale Verschiebung von ll (linker Kanal zum linken Kanal), lr (linker Kanal zum rechten Kanal), rr (rechter Kanal zum rechten Kanal), (linker Kanal zum rechten Kanal) an.

Sound.setVolume

```
Sound.setVolume(Wert);
```

Regelt die Lautstärke eines Sounds. Für `Wert` sind Werte zwischen 0 (stumm) und 100 (laut) zulässig.

Sound.start

```
Sound.start();
```

Startet die Wiedergabe eines Sounds.

Sound.stop

```
Sound.stop();
```

Stoppt die Wiedergabe eines Sounds.

_soundbuftime

```
_soundbuftime = Zahl;
```

Legt fest, wieviel Sekunden des Sounds gepuffert (vorgeladen) werden sollen. Voreinstellung ist hier 5 Sekunden.

splice

→ *Array.splice*

startDrag (Flash 4)

startDrag
→ 6.2 Bewegung – Einfluss durch den User und → 74 Soundeffekte.

```
startDrag ( Instanzname, Boolescher Wert,
relative Position)
```

Mit startDrag können Sie Instanzen an die Mauskoordinaten „heften“. Die relativen Positionen sind hierbei links, rechts, oben und unten. Hier geben Sie an, wo die Koordinaten refferenziert werden. Die Instanz wird so lange nachgeführt, bis sie mittels `stopDrag();` beendet wird: `startDrag("instanzname",true);`
→ MovieClip.startDrag

stop()

stoppt die Filmsequenz

stopAllSounds (Flash 4)

```
stopAllSounds();
```

Mit `stopAllSounds()` werden alle zur Zeit wiedergegebenen Sounds ausgesetzt. Erfüllt allerdings keine Mute-Funktion, neu angespielter Sound wird von Anfang an wiedergegeben.

stopDrag (Flash 4)

```
stopDrag();
```

Mittels `stopDrag` werden alle aktuellen Ziehoperationen beendet. *→ startDrag.*

String() (Flash 4)

```
String(Ausdruck);
```

Eine beliebige Zeichenfolge wird in einen String umgewandelt. Ausdruck kann hier z.B. eine Zahl sein, die Sie in eine vorhandene Zeichenkette eingliedern wollen. Die Variablentypen werden mit String kompatibel gemacht.

```
String(123)=x
Ergebnis: x="123"
```

String
→ 82 Externe Dateien einlesen.

String.charAt

```
Stringobj.charAt(index);
```

Liefert das Zeichen des `Stringobj` an der Position `index` zurück. Liegt der Index außerhalb des zulässigen Bereichs, wird eine leere Zeichenfolge zurückgegeben.

String.charCodeAt

```
Stringobj.charCodeAt(index);
```

Liefert den Wert eines Zeichens des `Stringobj` an der Position `index` zurück.

```
String.contact
Stringobj.contact(Wert1, Wert2, .. , Wertn)
```

Verbindet die Werte zu einem Stringobjekt und fügt diese an das Ende des vorhandenen an.

String.fromCharCode

```
Stringobj.fromCharCode(Chr1, Chr2, .. ,
Chrn)
```

Verbindet die Zeichen zu einem Stringobjekt und fügt diese an das Ende des vorhandenen an.

String.indexOf

```
Stringobj.indexOf(wert, start);
```

Gibt die erste Position einer gesuchten Zeichenfolge (`wert`) wieder. Die Suche startet im `Stringobj` an der Position `start`.

String.lastIndexOf

```
Stringobj.lastindexOf(wert, start);
```

Gibt die letzte Position einer gesuchten Zeichenfolge (`wert`) wieder. Die Suche startet im `Stringobj` an der Position `start`.

String.length

```
Stringobj.length
```

Gibt die Zahl der Zeichen in dem `Stringobj` zurück.

String.slice

```
Stringobj.slice(start, ende);
```

Extrahiert aus dem `Stringobj` eine Zeichenfolge von der Position `start` bis `ende`.

String.split
```
Stringobj.split(trennzeichen);
```
Zerlegt eine Zeichenfolge an den Stellen des angegebenen Trennzeichens. Wiedergegeben wird ein Array mit den Einzelsegmenten.

String.substr
```
Stringobj.substr(start, laenge);
```
Gibt aus dem `Stringobj` eine Zeichenfolge von der Position `start` an mit `laenge` Zeichen folgend.

String.substring
```
Stringobj.substring(start, ende);
```
Gibt aus dem `Stringobj` eine Zeichenfolge von der Position `start` bis `ende` zurück.

String.toLowerCase
```
Stringobj.toLowerCase();
```
Verwandelt das `Stringobj` komplett in Großbuchstaben.

String.toUpperCase
```
Stringobj.toUpperCase();
```
Verwandelt das `Stringobj` komplett in Kleinbuchstaben.

substring
```
substring( zeichenfolge, start, laenge);
```
Gibt aus der `zeichenfolge` eine neue Zeichenfolge von der Position `start` an mit `laenge` Zeichen folgend.

swapDepths
→ *MovieClip.swapDepths*

T

_target
```
instanzname._target
```
Gibt den Zielpfad der Instanz zurück.

targetPath
```
targetpath(filmsequenzobj);
```
Gibt den Zielpfad der Filmsequenz (z.B. `_root` oder `_parent`) zurück.

tellTarget
```
tellTarget(instanzname){
...
}
```
Führt die Anweisungen innerhalb des `telltarget` innerhalb der Instanz (`instanzname`) aus.

this
```
this
```
Verweist innerhalb eines Objektes auf dieses.

toggleHighQuality
```
toggleHighQuality();
```
Aktiviert und deaktiviert das Anti-Aliasing der Wiedergabe.

to String
→ *Array.toString*

_totalframes
```
instanzname._totalframes
```
Gibt die Anzahl der Frames der Instanz (`instanzname`) zurück.

totalframes,
→ *46 Preloader*
– Zusätze für Preloader.

trace
```
trace(ausdruck);
```
Gibt Werte des Ausdrucks (`ausdruck`) im Ausgabefenster des Filmtestmodus wieder.

typeof
```
typeof(ausdruck);
```
Wertet den Ausdruck aus, gibt zurück worum es sich dabei handelt. (Zeichenfolge, Filmsequenz, Objekt oder Funktion)

U

unescape

```
unescape(Ausdruck);
```

Umkehrfunktion zu `escape(Ausdruck)`.

unloadMovie

```
unloadMovie(Ebene);
```

→ *MovieClip.unloadMovie(Ebene).*

updateAfterEvent

```
updateAfterEvent(event);
```

Bei Eintreten des Events wird die Anzeige aktualisiert. Gültige Events sind:

- `mouseMove` – Bewegt sich die Maus?
- `mouseDown` – Ist die linke Maustaste gedrückt?
- `mouseUp` – Wird die linke Maustaste losgelassen?
- `keyDown` – Wird eine Taste gedrückt?
- `keyUp` – Wird eine Taste losgelassen?

_url

```
Filminstanz._url
```

Liefert als Wert die URL der .swf-Datei zurück.

V

var

```
var Variable;
```

Initialisiert Variablen.

_visible

```
Instanzname._visible
```

Liefert als Wert Boolean zurück, bzw. kann mittels Boolean (true / false) gesetzt werden, ob eine Instanz sichtbar ist (true) oder nicht (false).

void

```
void (Ausdruck);
```

Verwirft den Wert des Ausdruckes und liefert "undefined" zurück.

W

while

```
while (Bedingung) {
...
}
```

Führt Befehle innerhalb der geschweiften Klammern {} so lange aus, bis die *Bedingung* den Wert *false* hat (Boolean).

_width

```
Instanzname._width
```

Liefert als Wert die Breite der Instanz gemessen in Pixeln zurück, bzw. kann neu gesetzt werden. Die Proportionen werden nicht berücksichtigt.

Eigenschaften von Objekten

→ *6.2 Bewegung und folgende.*

with

```
with(object){
...
}
```

TellTarget-Nachfolger, → *tellTarget.*

X

_x

```
Instanzname._x
```

Eigenschaft, liest bzw. setzt die x-Koordinaten relativ zum übergeordneten Koordinatenursprung.

_xmouse

```
_xmouse
```

Eigenschaft, liest die x-Koordinaten des Cursors relativ zum übergeordneten Koordinatenursprung.

XML

```
new XML(Quelle);
```

Objekt, legt ein neues XML-Objekt an, kann optional auch eine bestehende XML-*Quelle* verwenden.

XML.appendChild

```
Vater.appendChild(Sohn);
```

Hängt einen neuen Knoten *Sohn* an den *Vater (XML-Objekt)*.

XML.attributes

```
Sohn.attributes.Eigenschaft ;
```

Liefert die *Eigenschaft* des *Sohnes (XML-Knoten)* zurück. Die Eigenschaft kann in XML frei benannt werden.

XML.childNodes

```
Vater.childNodes;
```

Liefert einen Array mit einer Verweisliste auf alle *Söhne (untergeordnete Knoten)* des *Vaters (XML-Knotens)* zurück.

XML.cloneNode

```
Sohn.cloneNode(Boolean);
```

Erzeugt einen Zwilling (Kopie) des Sohnes (XML-Knotens).

XML.createElement

```
Vater.createElement(neuerSohn);
```

Erzeugt einen neuen, untergeordneten Knoten.

XML.createTextNode

```
Gedicht.createTextNode(Vers);
```

Erzeugt einen dem Gedicht (XML-Objekt) untergeordneten Vers (neuen Knoten). Dieser Knoten beinhaltet Text.

XML.docTypeDecl

```
XMLobj.docTypeDecl;
```

Liest und schreibt die DOCTYPE-Deklaration des XML-Objektes.

XML.firstChild

```
Vater.firstChild;
```

Liefert den ersten Sohn (Knoten) des Vaters (übergeordnetes XML-Objekt) zurück.

XML.haschildNodes

```
Vater.haschildNodes();
```

Liefert Boolean (true oder false) zurück, inwieweit der Vater (XML-Knoten) Söhne (untergeordnete Knoten) hat.

XML.insertBefore

```
Vater.insertBefore(neuerSohn, Sohn);
```

Fügt einen neuen Sohn (untergeordneten Knoten) vor einen bestimmten anderen Sohn (untergeordneten Knoten) ein.

XML.lastChild

```
Vater.lastChild;
```

Liefert den letzten Sohn (Knoten) des Vaters (übergeordnetes XML-Objekt) zurück.

XML.load

```
XMLobj.load(URL);
```

Lädt ein neues XML-Objekt aus der URL. Alte Daten werden überschrieben.

XML.loaded

```
XMLobj.loaded;
```

Liefert Boolean zurück, ob das XMLobj vollständig geladen wurde.

XML.nextSibling

```
Sohn.nextSibling;
```

Liefert den Bruder (nächsten nebengeordneten Knoten) des Sohnes (XML-Knoten) zurück.

XML.nodeName

```
Vater.nodeName;
```

Liefert den Namen des Vaters (XML-Knoten) zurück. Falls es sich bei dem Knoten um einen Textknoten handelt, ist der Name „null“.

XML.nodeType

```
Knoten.nodeType;
```

Liefert den Typ des Knotens zurück.
1=XML-Knoten, 3=Textknoten

XML.nodeValue

```
Knoten.nodeValue
```

Liefert den Wert des Knotens zurück. Falls es sich bei dem Knoten um einen Textknoten handelt, wird der

Text wiedergegeben, ansonsten ist der Rückgabewert "null".

XML.onLoad

```
XMLobj.onLoad(Bool)
```

Wurde das XMLobj geladen, liefert diese Funktion für Bool den Booleschen Wert true zurück.

XML.parentNode

```
Sohn.parentNode;
```

Verweist auf den Vater (übergeordneten Knoten) des Sohnes (Knoten).

XML.parseXML

```
XMLobj.parseXML(Quelle);
```

Liest aus der *Quelle* das XML als Zeichenkette neu ein und verwirft das alte XMLobj.

XML.previousSibling

```
Sohn.previousSibling;
```

Liefert den Bruder (vorherigen nebengeordneten Knoten) des Sohnes (XML-Knoten) zurück.

XML.removeNode

```
Knoten.removeNode()
```

Entfernt den Knoten aus den übergeordneten Verweisen.

XML.send

```
XMLobj.send(URL, Ziel);
```

Versendet das *XMLobj* an die angegebene *URL*, im gewünschten *Ziel* (Fenster).

XML.sendAndLoad

```
XMLobj1.sendAndLoad(URL, XMLobj2);
```

Versendet das *XMLob1j* an die angegebene *URL* und lädt die Antwort des Servers in das *XMLobj2*.

XML.status

```
XMLobj.status;
```

Liefert den Status des XMLobj zurück.

0:	Keine Fehler
-2:	CDATA nicht richtig abgeschlossen
-3:	XML-Deklaration nicht richtig abgeschlossen
-4:	DOCTYPE-Deklaration nicht richtig abgeschlossen
-5:	Kommentar-Deklaration nicht richtig abgeschlossen
-6:	Ein XML-Element ist ungültig
-7:	Zu wenig Speicher
-8:	Ein Attribut ist nicht richtig abgeschlossen
-9:	Es gibt ein offenes Tag (Endtag fehlt)
-10:	Es gibt ein offenes Tag (Anfangstag fehlt)

XML.toString

```
Sohn.toString();
```

Liefert eine Zeichenkette mit allen Einträgen des Sohnes (XML-Knoten) zurück.

XML.xmlDecl

```
XMLobj.xmlDecl;
```

Liest und schreibt die Deklaration des XMLobj (XML-Objektes).

XML Socket

```
new XMLSocket();
```

Konstruktor, erstellt ein neues XML-Socket Objekt. Hält eine „offene" Verbindung zum Server, brauchbar für Multiplayer-Spiele oder Chat-Systeme.

XMLSocket.close

```
XMLSocket.close();
```

Beendet die Verbindung des XMLSocket.

XMLSocket.connect

```
XMLSocketobj.connect(Host, Port);
```

Baut eine XMLSocket –Verbindung mit dem Server auf. *Host* ist der DNS-Name oder die IP-Adresse, *Port* gibt die TCP-Protnummer an.

XMLSocket.onClose

```
XMLSocketobj.onClose();
```

Rückmeldungsfunktion, bei Beenden der Verbindung.

XMLSocket.onConnect

```
XMLSocketobj.onConnect(Bool);
```

Wird das XMLSocketobj verbunden, liefert diese Funktion für *Bool* den Booleschen Wert *true* zurück.

XMLSocket.onXML

```
XMLSocketobj.onXML(objekt);
```

Wenn das *objekt* die Verbindung passiert, können Befehle ausgeführt werden.

ZB:)

```
XMLSocketobj.onXML = kriteriumNode;
function kriteriumNode(doc){
var msg = doc.firstChild;
if (msg != null && doc.nodeName=="Gewünsch-
tesKriterium"){
...
     }
}
```

Sobald ein Knoten mit dem „*GewünschtesKriterium*“ den Socket passiert werden die Befehle innerhalb der Klammern ausgeführt.

XMLSocket.send

```
XMLSocketobj.send(object);
```

Versendet das *object* an den angebundenen Server.

_xmouse

```
Instanzname._xmouse;
```

Gibt den Wert der X-Cursorkoordinaten gemessen an dem Koordinatenursprung der Instanz *Instanzname* zurück

_xscale

```
Instanzname._xscale;
```

Liest, bzw. setzt die horizontale prozentuale Skalierung der Instanz Instanzname. Proportionen werden nicht berücksichtigt.

Eigenschaften von Objekten
→ *6.2 Bewegung und folgende.*

Y

_y

```
Instanzname._y
```

Eigenschaft, liest bzw. setzt die y-Koordinaten relativ zum übergeordneten Koordinatenursprung.

_ymouse

```
Instanzname._ymouse
```

Gibt den Wert der Y-Cursorkoordinaten gemessen an dem Koordinatenursprung der Instanz *Instanzname* zurück

_yscale

```
Instanzname._yscale;
```

Liest, bzw. setzt die vertikale prozentuale Skalierung der Instanz Instanzname. Proportionen werden nicht berücksichtigt.

Sonstige Syntax

-- (Dekrement)

```
--var1; var1--;
```

Eine Variable wird um den Wert 1 verringert. Dies geschieht mit einer Prä-Dekrement(--var1)- oder einer Post-Dekrement(var1--)-Operation.

```
var1 = 4;
var2 = --var1;
```

var2 und var1 würden den Wert 3 enthalten.

```
var1 = 4;
var2 = var1--;
```

var2 würde den Wert 4 enthalten, var1 den Wert 3. Möglich ist auch nur var1-- ohne Zuweisung, dabei würde var1 um 1 verringert.

++ (Inkrement)

```
++var1; var1++;
```

Ein Variable wird , um den Wert 1 erhöht. Dies geschieht mit einer Prä-Dekrement(++var1)- oder einer Post-Dekrement(var1+)-Operation.

```
var1 = 4;
var2 = ++var1;
```

var2 und var1 würden den Wert 5 enthalten.

```
var1 = 4;
var2 = var1++;
```

var2 würde den Wert 4 enthalten, var1 den Wert 5. Möglich ist auch nur var1++ ohne Zuweisung, dabei würde var1 um 1 erhöht.

! (logisches NICHT)

```
!var1
```

Kehrt den Boolschen Wert einer Variable oder eines Ausdruckes um.

True wird zu false und
False wird zu true

```
var1 = false;
If(!var1){
Trace("False wird zu true");
}
```

Diese If-Bedinung würde wahr werden.

!= (nicht gleich)

```
var1 != var2
```

Vergleicht zwei Variablen und liefert true zurück, wenn diese ungleich sind. Es ist genau das Gegenteil von dem == - Operator.

```
var1="Hallo";
var2=" Welt";
If (var != var2){
Trace("var1 unterscheidet sich von var2");
}
```

% (Modulo)

var1 % var2

Teilt die erste Variable durch die zweite und gibt als Ergebnis den Rest an.

13 % 4 ergibt 1.
15 % 4 ergibt 3.

%= (Modulo-Zuweisung)

```
var1 %= var2
```

Entspricht der Modulo-Anweisung, nur dass zusätzlich der Wert direkt der ersten Variable zugewiesen wird. Dies ist dementsprechend eine verkürzte Schreibweise.

```
var1 %= var2
```

entspricht

```
var1 = var1 % var2
```

& (Bitweises UND)

```
var1 & var2
```

Wandelt var1 und var2 in vorzeichenlose 32-Bit-Ganzzahlen um und führt eine Boolean-UND-Operation für jedes einzelne Bit der ganzzahligen Argumente aus. Das Ergebnis ist eine neue vorzeichenlose 32-Bit-Ganzzahl.

Im Flash-4-Player wurde dieser Befehl zum Verketten von Variablen und Stringnamen benutzt. In Flash 5 wurde dieser durch `add` und + ersetzt.

&& (kurzgeschlossenes UND, logisches UND)

```
var1 && var2
```

führt einen logischen Vergleich der beiden Variablen oder Ausdrücke durch. Nur wenn var1 und var2 „true“ sind, ergibt der Vergleich auch ein true.

Logisches „Und“ kann auch durch and ersetzt werden.

Wert1 && Wert2 oder Wert1 and Wert2

Wert 1	Wert 2	Ergebnis
true	true	true
true	false	false
false	true	false
false	false	false

&= (bitweise UND-Zuweisung)

```
var1 &= var2
```

Dies ist eine verkürzte Schreibweise für das Bitweise UND &. Neben de, Bitweisen-Vergleich wird das Ergebnis dabei sofort der ersten Variable zugewiesen.

```
var1 &= var2;
```

entspricht `var1 = var1 & var2`

( ) (Klammern)

Dient zum Zusammenfassen von bevorzugten Operationen, z.B. um andere Operationen nachzustellen.

```
var1 * var2 + var3;
var1 * (var2 + var3);
```

Dient bei Funktionen zum Übergeben von Argumenten.

```
return(var1);
```

- (Minus)

```
-var1
```

Dient zu Negation oder Subtraktion. Als Negator kehrt er das Vorzeichen des Wertes um.

```
-(2+3) ergibt -5
```

Bei der Subtraktion wird der zweite Wert vom ersten subtrahiert.

```
5 - 2 ergibt 3
```

*** (Multiplikation)**

```
var1 * var2
```

Dient zum Multiplizeiren von zwei Werten

```
5 * 8 ergibt 40.
```

***= (Multiplikationszuweisung)**

Dies ist eine verkürzte Schreibweise für die Multiplikations-Operation (*). Weist das Ergebnis dem ersten Wert zu.

```
var1=5;
var1 *=8;
```

var1 würde den Wert 40 enthalten.

, (Komma)

Trennt mehrere Varibalen/Werte, um diese mit derselben vorherigen Operation zu benutzen.

Funktionsaufruf(wert1, wert2, wert3)

Oder

```
var var1=1, var2=1, var3=1;
```

wäre gleichbedeutend mit

```
var var1=1;
var var2=1;
var var3=1;
```

→ *. (Punkt–Operator)*

Mit der Punkt-Syntax kann man auf andere Movies direkt zugreifen.

```
_root.instanzname.var1=5;
```

Hier würde die Variable var1 in dem Movie mit dem Namen Instanzname im Hauptverzeichnis _root gleich 5 gesetzt.

Durch den Punkt-Sytax ist es auch der Befehl `getProperty` überflüssig geworden, da man direkt auf die Eigenschaften zugreifen kann. Z.B. auf die y-Position des Movies.

```
root.Movie._y
```

Punktsyntax
→ *Kapitel 1: Einführung und Kapitel 2: Neuerungen in Flash5.*

?: (bedingt)

Wert = Bedingung? Zuweisungs-variable/wert bei True : Zuweisungs-variable/wert bei false

Dies ist eine verkürzte Schreibweise für eine If-Abfrage

```
If( Wert > 5){
x=1;
} else {
x=2;
}
```

Statt dieser langen if-Abfrage kann man auch das Folgende schreiben.

```
x = Wert>5 ? 1 : 2;
```

/ (Division)

```
var1 / var2
```

Teilt den ersten Wert durch den zweiten Wert.

```
6 / 2 ergibt 3.
```

// (kommentartrennzeichen)

```
// Dies ist ein Komentar
```

Sorgt dafür das der in der Zeile stehende Text nicht von Flash bearbeitet wird und gibt einem dadurch die Möglichkeit, Kommentare in den Quelltext zu schreiben.

/* */(Kommentartrennzeichen)

```
/* Dies ist ein Kommentar der auch
durch aus über mehrere Zeilen gehen kann */
```

Der Text zwischen den zwei Zeilen wird als Kommentar erkannt.

/= (Divisionszuweisung)

```
var1 = 6;
var1 /= 2;
```

var1 wird durch 2 geteilt und das Ergebnis zugeordnet, so dass diese später den Wert 3 enthält. Dies ist eine verkürzte Schreibweise für /.

Dies entspricht

var1 = var1 / 2;

[ ] (Arrayzugriffsoperator)

```
NamedesArray[ ];
Objekt[wert1, wert2, ...wertN]
```

Erstellt ein neues Objekt und initialisiert die in den Argumenten angegebenen Eigenschaften bzw. initialisiert ein neues Array mit den Elementen, die in den Argumenten angegeben werden.

^(Bitweises XODER)

```
var1 ^ var2
```

Wandelt var1 und var2 in vorzeichenlose-32-Bit-Ganzzahlen um und gibt an jeder Bit-Position eine 1 zurück, an der die entsprechenden Bit-Ganzzahlen in var1 oder var2 (jedoch nicht in beiden) 1 sind.

^= (Bitweise XODER – Zuweisung)

```
var1 ^= var2;
```

Dies ist eine verkürzte Schreibweise für ^ (Bitweise XODER). Hierbei wird die Operation ^ (Bitweise XODER) mit den Variablen var1 und var2 ausgeführt und das Ergebnis var1 zugewiesen. Dies entspricht der Schreibweise

```
var1 = var1 ^ var2;
```

{ } (Objektinitialisierung)

```
objekt { var1: wert, var2: wert2, var3
[wert3, wert4, wert5]
```

Erstellt ein neues Objekt und initialisiert es mit den festgelegten Variablen und Werten. Die Verwendung dieses Operators entspricht dem Aufruf von `new Objekt` und der anschließenden Zuweisung von Variablen und Werten. Diese Operation kann anstelle des `new`-Operators verwendet werden und bietet Ihnen eine schnelle und einfache Alternative der Objekterstellung.

```
Buch { name: "Flash 5 - Actionscript"
  autoren : [" Jan Brücher" , "Marc Hugo"]
};
```

Es wurde ein Objekt mit dem Namen *Buch* erstellt, welches die Variable *name* und ein Array *autoren* enthält.

| (Bitweises ODER)

```
var1 | var2
```

Wandelt var1 und var2 in Vorzeichenlose 32-Bit-Ganzzahlen um und gibt an jeder Bit-Position eine 1 zurück, falls dort eine 1 bei var1 oder var2 vorhanden ist, gemäß ODER-Logik.

8 | 5 ergibt 13
(1000 | 0101 = 1101)
8 | 13 ergibt 13
(1000 | 1101 = 1101)

|| (ODER)

```
var1 || var2
```

Ein Boolscher Wert oder Ausdruck, der in einen Boolescher Wert umgewandelt wird.

Bei Ausdrücken, die nicht vom Typ Boolean sind, weist der logische ODER-Operator Flash an, den linken Ausdruck auszuwerten.

Wert1 || Wert2 oder Wert1 or Wert2

Wert 1	Wert 2	Ergebnis
true	true	true
true	false	true
false	true	true
false	false	false

|= (Bitweise ODER-Zuweisung)

```
var1 |= var2
```

Dies ist eine verkürzte Schreibweise für | .

Es wird var1 | var2 berechnet und das Ergebnis var1 zugewiesen.

```
var1=10;
var2=13;
var1 |= var2;
var1 = var1 | var2;
```

1111= 1010 | 1101
var1 = 1111
var1 würde den Wert 15 enthalten.

~ (Bitweises NICHT)

```
~var
```

Wandelt var1 in eine vorzeichenlose 32-Bit-Ganzzahl um und invertiert die Bits dann. Das bedeutet, dass die Zahl um eins erhöht wird und das Vorzeichen sich ändert.

Aus 1 wird –2.
Aus 2 wird -3.

Aus 3 wird –4.
Aus –5 wird 4.
Aus –6 wird 5.
Usw.

+ (Addition)

```
var1 + var2
```

Addiert numerische Werte oder verkettet Zeichenfolgen. Wenn ein Ausdruck eine Zeichenfolge ist, werden alle Ausdrücke in Zeichenfolgen umgewandelt und verkettet.

```
var1 = 4;
var2 = 7;
var3 = "Hallo Welt ";
var1 + var2 ergibt den Wert 11.
var3 + var2 ergibt den String „Hallo Welt 7"
var3 +var2 +var1 ergibt den String „Hallo
Welt 74"
```

+= (Additionszuweisung)

```
var1 += var2
```

Dies ist eine verkürzte Schreibweise für + .

Addiert var1 und var2 und weist das Ergebnis var1 zu. Falls var1 oder var2 ein String ist, wird var2 an var1 angehängt und var1 würde danach den verketteten String von var1 + var2 enthalten.

```
var = 1;
var2 = 3;
var1 += var2;
```

var1 enthält den Wert 4.

```
var1 = var1 + var2;
```

< (kleiner als)

```
var1 < var2
```

Vergleicht zwei Ausdrücke oder Variablen, ermittelt, ob var1 kleiner ist als var2 ist, und gibt dann true aus, falls dies zu trifft.

```
Bruder = 5;
Schwester = 10;
If ( Bruder < Schwester){
trace("Der Bruder ist jünger als die
Schwester");
}
```

<< (Bitweises Shift links)

```
var1<<var2
```

var1 wird bitweise um die Anzahl in var2 nach links verschoben. Var2 muss eine ganze Zahl sein und zwischen 0 und 31 liegen. Die leeren Bitpositionen werden mit 0 aufgefüllt. Das Verschieben des Werts nach links entspricht der jeweiligen Multiplikation mit 2.

```
var1 = 1 << 10;
```

Das Ergebnis dieser Operation lautet var1 = 1024. Dies berechnet sich wie folgt: 1 dezimal ist gleich 0001 binär, 0001 binär um 10 nach links verschoben ist 10000000000 binär, und 10000000000 binär ist 1024 dezimal.

```
var1 = 7 << 8;
```

Das Ergebnis dieser Operation lautet var1 = 1792. Dies berechnet sich wie folgt: 7 dezimal ist 0111 binär, 0111 binär um 8 nach links verschoben ist 11100000000 binär, und 11100000000 binär ist 1792 dezimal.

<<= (Bitweises Shift links und Zuweisung)

```
var1 <<= var2
```

Dies ist eine verkürzte Schreibweise für <<.

var1 wird um die in var2 angegebene ganze Zahl zwischen 0 und 31 bitweise nach links verschoben, dass Ergebnis wird in var1 gespeichert.

```
var1 <<= var2;
```

Dies entspricht dem folgenden Befehl.

```
var1 = var1 << var2;
```

<= (kleiner oder gleich)

```
var1 <= var2
```

Vergleicht zwei Ausdrücke oder Variablen, ermittelt, ob var1 kleiner oder genauso groß wie var2 ist, und gibt dann true aus, falls dies zu trifft.

```
Bruder = 5;
Schwester = 10;
If ( Bruder <= Schwester){
trace("Der Bruder ist nicht älter als die
Schwester");
}
```

<> (nicht gleich)

```
var1 <> var2
```

Vergleicht zwei Ausdrücke oder Variablen, ermittelt, ob var1 ungleich var2 ist, und gibt dann true aus, falls dies zu trifft.

```
Bruder = 5;
Schwester = 10;
If ( Bruder <> Schwester){
trace("Der Bruder und die Schwester sind
nicht gleich alt.");
}
```

Diese Funktion gilt in Flash5 als veraltet. Stattdessen wird die Verwendung des neuen !=-Operators empfohlen.

= (Zuweisung)

```
var1= var2
```

Weist der Variablen var1 den Wert aus var2 zu.

```
var1 = "Hallo Welt";
```

var1 hat nun den String „Hallo Welt“

```
var2 = var1;
```

var2 hat nun auch den String „Hallo Welt“.

-= (Negationszuweisung)

```
var1 -= var2
```

Dies ist eine verkürzte Schreibweise für - .

Von var1 wird der Wert in var2 abgezogen und das Ergebnis wird var1 zugeordnet.

```
var1 -= var2;
```

dies entspricht folgender Schreibweise:

```
var1 = var1 - var2;
```

== (gleich, Vergleichsoperator)

```
var1 == var2
```

Prüft zwei Ausdrücke auf Gleichheit. Das Ergebnis ist true, wenn die Ausdrücke gleich sind.

```
Bruder = 5;
Schwester = 10;
If ( Bruder == Schwester){
trace("Der Bruder und die Schwester sind
gleich alt.");
}
```

> (größer als)

```
var1 > var2
```

Vergleicht zwei Ausdrücke oder Variablen, ermittelt, ob var1 größer ist als var2 ist, und gibt dann true aus, falls dies zu trifft.

```
Bruder = 5;
Schwester = 10;
If ( Bruder > Schwester){
trace("Der Bruder ist älter als die Schwes-
ter");
}
```

>= (größer oder gleich)

```
var1 >= var2
```

Vergleicht zwei Ausdrücke oder Variablen, ermittelt, ob var1 größer oder genauso groß wie var2 ist, und gibt dann true aus, falls dies zu trifft.

```
Bruder = 5;
Schwester = 10;
If ( Bruder >= Schwester){
trace("Die Schwester ist jünger als der
Bruder");
}
```

>> (Bitweises Shift rechts)

```
var1>>var2
```

var1 wird bitweise um die Anzahl in var2 nach rechts verschoben. Var2 muss eine ganze Zahl sein und zwischen 0 und 31 liegen. Die leeren Bitpositionen werden mit 0 aufgefüllt. Das Verschieben des Werts nach rechts entspricht der jeweiligen Division mit 2.

```
var1 = 65535 >> 8;
```

var1 enthält dann die Zahl 255.

Dies berechnet sich wie folgt: 65535 dezimal ist gleich 1111111111111111 binär, 1111111111111111 binär um acht Bit nach rechts verschoben ist 11111111 binär und dies ist 255 dezimal.

>>= (Bitweise Shift rechts und Zuweisung)

```
var1 >>= var2
```

Dies ist eine verkürzte Schreibweise für >>.

var1 wird um die in var2 angegebene ganze Zahl zwischen 0 und 31 bitweise nach rechts verschoben, dass Ergebnis wird in var1 gespeichert.

```
var1 >>= var2;
```

Dies entspricht dem folgenden Befehl:

```
var1 = (var1 >> var2);
```

>>> (Vorzeichenloses bitweises Shift rechts)

```
var1 >>> var2
```

Entspricht dem Operator für bitweises Shift rechts (>>>), bewahrt jedoch nicht das Vorzeichen von var1, da die Bits auf der linken Seite stets mit o aufgeführt werden.

```
var1 = -1 >>> 1;
```

var1 enthält anschließend den Wert 2147483647.

Dies berechnet sich wie folgt: -1 dezimal ist gleich 11111111111111111111111111111111 binär, beim vorzeichenlosen Verschieben um ein Bit nach rechts wird das erste Bit (rechts außen) entfernt und das letzte vorzeichenrelevante Bit (links außen) mit o aufgefüllt. Das Ergebnis lautet wie folgt: 01111111111111111111111111111111 binär, was für die 32-Bit-Ganzzahl 2147483647 steht.

>>>= (Vorzeichenloses bitweises Shift rechts und Zuweisung)

```
var1 >>>= var2
```

Dies ist eine verkürzte Schreibweise für >>>.

Es wird eine vorzeichenlose bitweise Shift-rechts-Operation durchgeführt, wobei das Ergebnis in var1 gespeichert wird.

```
var1 >>>= var2;
```

Dies entspricht dem folgenden Befehl:

```
var1 = (var1 >>> var2);
```

Pearson Education

ADDISON-WESLEY

Anhang A

Tastaturcode

Buchstaben von A bis Z und Ziffern 0 bis 9:

Taste	Code
A	65
B	66
C	67
D	68
E	69
F	70
G	71
H	72
I	73
J	74
K	75
L	76
M	77
N	78
O	79
P	80
Q	81
R	82
S	83
T	84
U	85
V	86
W	87
X	88
Y	89
Z	90
0	48
1	49
2	50
3	51
4	52
5	53
6	54
7	55
8	56
9	57

Tasten auf dem numerischen Ziffernblock:

Tasten	Code
0	96
1	97
2	98
3	99
4	100
5	101
6	102
7	103
8	104
9	105
*	106
+	107
↵	108
-	109
,	110
/	111

Funktionstasten:

Tasten	Code
F1	112
F2	113
F3	114
F4	115
F5	116
F6	117
F7	118
F8	119
F9	120
F10	121
F11	122
F12	123

Andere Tasten:

Tasten	Code
←	8
⇆	9
Entf	12
↵	13
⇧	16
Strg	17
Alt	18
⇩	20
Esc	27
Leertaste	32
Bild ↑	33
Bild ↓	34
Ende	35
Pos1	36
←	37
↑	38
→	39
↓	40
Einfg	45
Rückschritt	46
Hilfe	47
Num	144
;:	186
=+	187
-_	189
/?	191
`-	192
[{	219
|	220
]}	221
“’	222

CD-ROM/Website

Auf der CD-ROM finden Sie zu jedem im Buch erwähnten Beispiel das passende File. Zudem enthält die CD-ROM Tryout-Versionen ausgewählter Tools, die Flash hervorragend ergänzen.

Sie können uns im Internet auf der Seite *www.DieFlasher.de* besuchen. Auf dieser Website gibt es einen gesonderten Bereich speziell für die Käufer dieses Buches. Mit der untenstehenden ID-Nummer haben Sie Zugang zu diesem Bereich. Hier können Sie Aktualisierungen, Verbesserungen und zusätzliche Informationen abrufen. Für den Fall, dass noch die eine oder andere Frage offen geblieben ist, haben wir dort eigens auch ein Forum eingerichtet.

ID-Nummer: 1676-4583-2134

Aktuelle Informationen zu weiteren Publikationen finden Sie unter *http://www.addison-wesley.de*.

Pearson Education

ADDISON-WESLEY

Index