



# Workshop: Director FOLGE 14

## Digitales Fotoalbum

In der letzten Folge unseres Workshops produzieren wir einen Film, der beliebig viele Fotos mit Titel und beschreibendem Text speichert und so als Grundlage für digitale Alben oder Kataloge dienen kann.



■ Der vorliegende Director-Film beschränkt sich aus Platzgründen auf wenige, erweiterbare Funktionen (Abbildung 1). Die darin enthaltenen Bilder und Texte lassen sich sowohl der Reihe nach als auch mit einer einfachen Suchfunktion aufrufen. Das Scripting der Navigations- und Suchfunktionen setzt einerseits eindeutige Dateipfade und andererseits eindeutige Bildtitel voraus. Es ist also nicht möglich, ein Bild mehrfach mit verschiedenen Titeln zu laden oder unterschiedlichen Bildern den gleichen Titel zu geben.

**Speicherung der Bilder.** Da sich die Bilder auch hinzufügen und löschen lassen sollen, wenn der fertige Film von einem Projektor aus gestartet wird, bietet sich anstelle des Standard-Imports die externe Speicherung der Bilder an: Im Film selbst

werden also nur Verweise auf die externen Dateien gesichert. Dies bringt unter anderem den Vorteil, dass der benötigte Speicherbedarf des Films mit zunehmender Bildanzahl nur geringfügig wächst.

Im Film selbst speichern wir nur einen Platzhalter, den Bitmap-Darsteller namens „Dummy“, und verwenden ihn zur Anzeige der externen Bilddateien, indem wir die Darstellerproperty „the filename of member“ setzen, mit der sich auch zur Laufzeit des Programms Bezüge zu externen Daten herstellen oder editieren lassen.

**Der Button „Bild laden“.** Der Film basiert auf der Verwendung zweier Listen, die wir in globalen Variablen speichern, damit alle Prozeduren des Films sie verwenden können. Die lineare Liste „gImageList“ enthält die

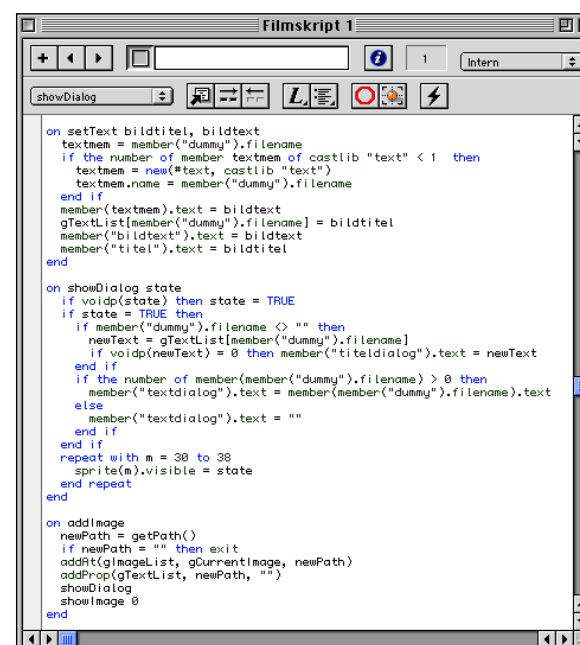
vollständigen Pfadangaben aller in den Film geladenen Bilddateien. In diese Liste gelangen die Pfadangaben mithilfe der Prozedur „addImage“ (Abbildung 2), die das Behavior des Buttons „Bild laden“ (Abbildung 3) aufruft. AddImage verwendet die Prozedur „getPath()“, die mithilfe des Xtras „FileIO“ einen Datei-öffnen-Dialog generiert und den Pfad der vom Anwender ausgewählten Datei als Ergebnis liefert. Sofern im Dialog nicht auf „Abbrechen“ geklickt wurde (in diesem Fall ist das Ergebnis des Aufrufs eine leere Zeichenkette), wird die Pfadangabe mit dem Befehl „add“ in die globale Bilderliste „gImageList“ eingetragen und der Dialog zur Eingabe des Bildtitels sowie des beschreibenden Textes angezeigt.

Die zweite Liste „gTextList“ ist eine Eigenschafts- beziehungsweise Propertyliste und enthält die Verknüp-

### 1: Das fertige Fotoalbum



### 2: Die Prozedur „addImage“



fung zwischen den Dateipfaden und den vom Anwender im Textdialog eingegebenen Bildtiteln. Den jedes Bild beschreibenden Fließtext speichern wir in Textdarstellern, die der Einfachheit halber den Namen des jeweiligen Dateipfads tragen.

Wenn der Anwender beispielsweise die im Ordner „Mein Mac:Daten:Fotos:“ gespeicherte Datei „Foto1.jpeg“ lädt und mit dem Bildtitel „Mein erstes Foto im Jahr 2000“ versieht, speichert die Bildliste „gImageList“ lediglich den Dateipfad zu diesem Ordner, also „[Mein Mac:Daten:Fotos:Foto1.jpeg]“.

Die Textliste „gTextList“ nimmt den Namen des neu erzeugten Textdarstellers als Eigenschaft sowie den eingegebenen Titel als Wert auf, in diesem Fall also „[Mein Mac:Daten:Fotos: Foto1.jpeg]:Mein erstes Foto im Jahr 2000]“.

Im Textdarsteller „Mein Mac:Daten:Fotos:Foto1.jpeg“ speichern wir den das Bild beschreibenden Fließtext (Abbildung 4). Um die vom Programm dynamisch erzeugten und auch wieder gelöschten Textdarsteller weitgehend von den anderen Darstellern des Films zu trennen, werden diese in einer eigenen, externen Besetzung verwaltet.

**Der Textdialog.** Sobald ein neues Bild geladen wurde, zeigt Director automatisch den Textdialog an, der sich später auch mithilfe des Buttons „Ti-

tel/Text editieren“ aufrufen lässt (Abbildung 5). Die Generierung der Textdarsteller und das Speichern des Textes übernimmt die im Filmskript gespeicherte Prozedur „setText“. Diese wird vom Behavior des Buttons „OK“ des Textdialogs (Abbildung 6) aufgerufen, das die beiden Parameter „Bildtitel“ und „Bildtext“ übergibt. Zuerst ermitteln wir hier den Dateipfad des aktuellen Bildes, indem wir die Darstellerproperty „the filename“ des Darstellers „Dummy“ auslesen. Für den Fall, dass noch kein Textdarsteller gleichen Namens in der Besetzung „text“ existiert, erzeugen wir diesen mithilfe der Funktion „new()“ und benennen ihn entsprechend (siehe Abbildung 2).

Die globale Variable „gCurrentImage“ verwenden wir als Zeiger auf das aktuelle Bild in der Bildliste „gImageList“. Der Wert von „gCurrentImage“ wird zum Beispiel von den Behaviors der Zurück- und Weiter-Buttons modifiziert. Beide Behaviors rufen die im Filmskript gespeicherte Prozedur „showImage“ auf (Abbildung 7) und unterscheiden sich nur in der als Parameter übergebenen Richtung (1 beziehungsweise -1). Nach der Überprüfung der Bildliste addiert diese Prozedur den als „direction“ übergebenen Richtungsparameter zum aktuellen Wert von „gCurrentImage“ und testet daran anschließend, ob der Anfang oder das Ende der Bildliste überschritten wur-

de. In diesen Fällen wird „gCurrentImage“ wieder auf das Ende respektive den Anfang der Liste gesetzt, sodass man vom Anfang der Liste auch direkt an deren Ende blättern kann – und umgekehrt. Mithilfe der Variablen „gCurrentImage“ ist es weiterhin möglich, neue Bilder an jeder beliebigen Position der Bildliste hinzuzufügen: Da Propertylisten den dazu verwendeten Befehl „addAt“ nicht unterstützen, ist die Reihenfolge der Einträge in der Textliste „gTextList“ nicht mit der Reihenfolge in „gImageList“ identisch, sodass wir nicht einfach an der mit „gCurrentImage“ adressierten Position auf „gTextList“ zugreifen können. Stattdessen suchen wir die Position des aktuellen Eintrags in der Textliste mit der Funktion „getPos()“. Die Anzeige des Bildtitels erreichen wir, indem wir den zum aktuellen Dateipfad gehörenden Wert in der Textliste auslesen und den Wert in den Textdarsteller „titel“ schreiben:

```
titel = gTextList[member("dummy").filename]
member("titel").text = titel
```

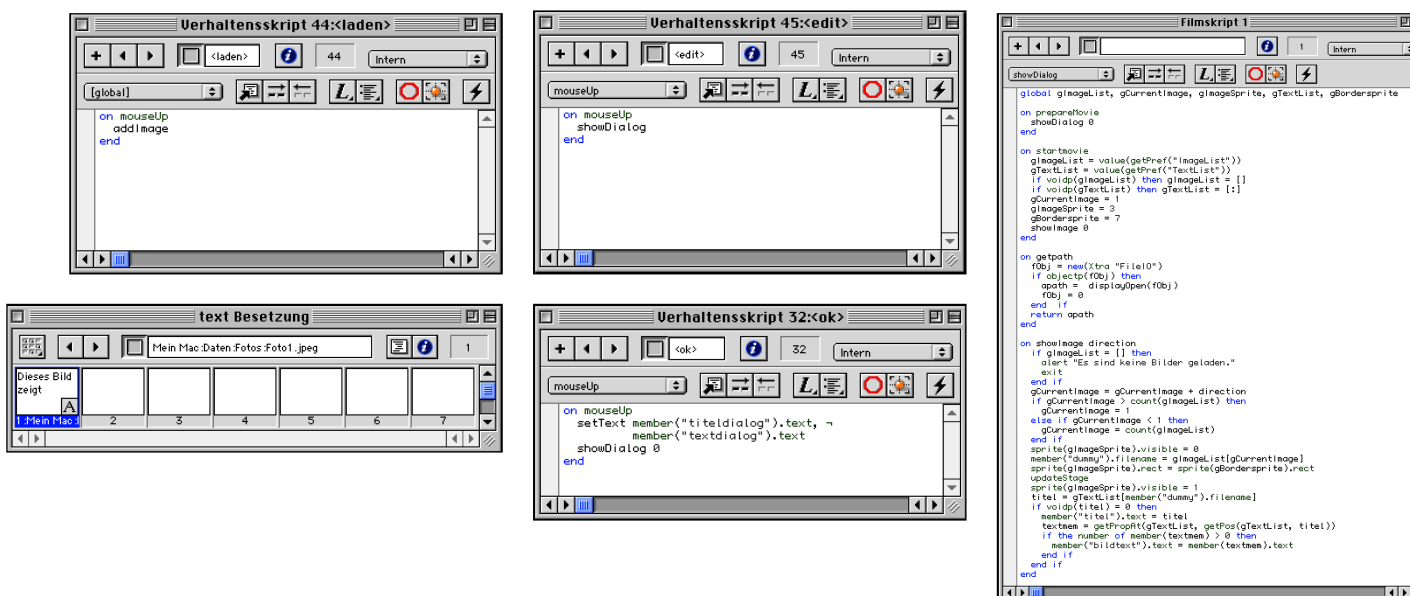
Den beschreibenden Fließtext enthält der mit dem Dateipfad bezeichnete Textdarsteller, im vorliegenden Fall also:

```
textmem = getPropAt(gTextList,
getPos(gTextList, titel)) →
```

### 3 + 4: Verhaltensskript, Textdarsteller

### 5 + 6: Textdialog generieren

### 7: Prozedur „showImage“



→ Wir ermitteln zunächst die Position des aktuellen Bildtitels in der Liste („getPos()“) und lesen dann an der entsprechenden Stelle die Eigenschaft aus („getPropAt()“).

Die Anzeige des Dialogs zur Texteingabe lässt sich durch die Platzierung der entsprechenden Darsteller in den obersten vom Film verwendeten

eigenständige Prozedur aus und speichern diese ebenfalls im Filmskript. Die Prozedur „showDialog“ bestimmt anhand des beim Aufruf übergebenen Parameters „state“, ob die vom Textdialog verwendeten Kanäle 30 bis 38 ein- oder ausgeschaltet werden sollen. Wenn kein Parameter angegeben wird, der Parameter „state“ also den unbestimmten Wert „<void>“ hat, schalten wir den Dialog per Default ein. Die Verwendung des Parameters „state“ hat den Vorteil, dass sich für Ein- und Ausblendung ein und dasselbe Skript verwenden lässt, was neben einer eventuell notwendigen Fehlersuche auch spätere Änderungen und Erweiterungen erleichtert. Für das Ausschalten des Dialogs vor dem Start der Wiedergabe sorgt der Eintrag „showDialog 0“ in der Prozedur „prepareMovie“ des Filmskripts. Das Behavior des Dialog-Buttons „Abbrechen“ enthält denselben Aufruf (Abbildung 8).

Sobald der Textdialog erscheint, sollen keine Mausclicks auf die hinter dem Dialog liegenden Sprites mehr möglich sein – der Dialog soll sich

modal verhalten. Dies erreichen wir, indem wir ein unsichtbares QuickDraw-Rechteck im Kanal 30 platzieren, das die gesamte Bühne abdeckt und alle relevanten Mausereignisse (hier „mouseDown“ und „mouseUp“) stoppt – durch leere, zum Beispiel nur aus einem Kommentar bestehende Prozeduren (Abbildung 9). Da das im Kanal 30 platzierte Rechteck natürlich auch im Stopp-Modus die Markierung der Sprites in den unteren Kanälen verhindert, müssen Sie diesen Kanal während der Arbeiten am Film manuell abschalten: Den entsprechenden Button finden Sie im Drehbuch vor der Spalte mit den Kanalnummern.

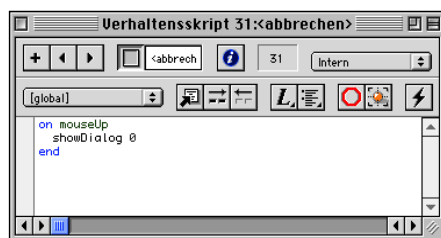
**Die Navigation.** Zum Blättern in den einzelnen Bildern und Texten verwenden wir zwei Buttons, die den jeweils vorhergehenden beziehungsweise den folgenden Eintrag anzeigen, indem sie die im Abschnitt „Der Textdialog“ weiter oben (siehe Abbildung 7) bereits vorgestellte Prozedur „showImage“ aufrufen (Abbildungen 10 und 11). →

## Die Anwendung soll es ermöglichen, den Textdialog von verschiedenen Stellen ein- und wieder auszublenden

ten Kanälen realisieren. Sobald der Dialog benötigt wird, schalten wir die Anzeige der Kanäle ein, setzen also die Sprite-Property „the visible of sprite n“ beziehungsweise „sprite(n).visible“ auf den Wert TRUE.

Da es immer auch möglich sein soll, den Dialog von verschiedenen Stellen der Anwendung aus ein- und wieder auszublenden, lagern wir die notwendigen Skriptzeilen in eine

8: Der Befehl „showDialog 0“



9: „Leere“ Prozedur



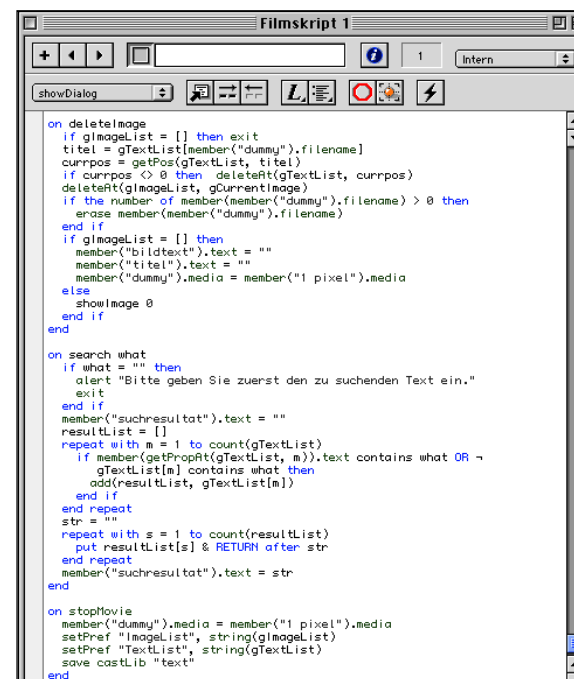
10: Button zum Weiterblättern



11: Button zum Zurückblättern



12: Suchfunktion



→ **Die Suchfunktion.** Mithilfe der Suchfunktion lassen sich in der rechts angeordneten Textbox all jene Einträge anzeigen, deren Text den eingegebenen Suchkriterien entspricht. Das Programm berücksichtigt dabei sowohl den Titel selbst als auch den beschreibenden Text. Die Angabe mehrerer Zeichenketten in einer Abfrage und deren logische Verknüpfung unterstützt die Suchfunktion in der vorliegenden Fassung nicht; der Einsatz einer entsprechenden Erweiterung ist bei dem gewählten Lösungsansatz aber prinzipiell möglich.

### Die Suchfunktion wird so programmiert, dass sie sowohl die Titel der Bilder als auch die beschreibenden Texte durchforstet

Die Suchfunktion besteht aus der im Filmskript gespeicherten Prozedur „search“ (Abbildung 12). Dieser Prozedur wird beim Aufruf der zu suchende Text im Parameter „what“ übergeben. In unserem Beispiel verwenden wir den Text, der in den als „bearbeitbar“ deklarierten Textdarsteller „suchtext“ eingegeben wurde, als Parameter, also die Darstellerproperty „the text of member „suchtext““ respektive in Punktyntax „member („suchtext“).text“.

Die Suche wird durch zwei verschiedene Ereignisse gestartet. Sowohl ein Mausklick auf den Button „suchen“ (Abbildung 13) als auch ein Druck auf die Eingabetaste ruft die Prozedur „search“ auf. Zur Erkennung der Eingabetaste verwenden wir das Behavior des Eingabefelds „suchtext“ (Abbildung 14). Das Behavior enthält die Prozedur „keyUp“,

die Director immer dann interpretiert, wenn der Anwender eine Taste losgelassen hat. Beachten Sie jedoch, dass Sprites dieses Ereignis nur dann empfangen können, wenn sie mit einem bearbeitbaren Textdarsteller verknüpft sind.

Die Prozedur „search“ arbeitet nach dem folgenden einfachen Prinzip: Zunächst prüft sie, ob überhaupt ein zu suchender Text übergeben wurde. Im Fehlerfall beenden wir die Suche hier mit einem entsprechenden Dialog. Dann löscht die Prozedur die Anzeige der Suchergebnisse im Darsteller „Suchresultat“ und legt die zur Speicherung der Suchergebnisse verwendete lineare Liste „resultList“ an. Die folgende Repeat-Schleife wird für alle Einträge der globalen Textliste „gTextList“ durchlaufen. Sie prüft mithilfe des Zeichenkettenoperators „contains“, ob der zu suchende Text in den Textdarstellern enthalten ist, auf die unsere Liste verweist, oder ob der Suchtext in den Bildtiteln selbst, also den Werten der Liste „gTextList“, vorkommt. Wir verwenden hier die logische Verknüpfung „OR“: Sobald eines der beiden Kriterien erfüllt ist, wird der entsprechende Eintrag der Textliste unserer Ergebnisliste „resultList“ hinzugefügt. In der anschließenden Repeat-Schleife bilden wir die zur Anzeige des Suchergebnisses verwendete Zeichenkette „str“. Diese lokale Variable wird außerhalb der Schleife zunächst mit einer leeren Zeichenkette initialisiert. In jedem Durchlauf der Schleife schreiben wir dann ein Element der Ergebnisliste inklusive eines Zeilenwechsels an das Ende der Zeichenkette. Das Resultat dieser Aktion zeigen wir auf der Bühne an, indem wir es in die Textkomponente des Darstellers „Suchresultat“ übertragen.

**Das Behavior der Ergebnisliste.** Ein Klick auf eine der Zeilen des Textdarstellers „Suchresultat“ soll natürlich das entsprechende Bild nebst zugehörigem Text anzeigen. Zu diesem Zweck muss das Behavior (Abbildung 15) auf die beiden globalen Listen sowie auf das aktuelle Bild „gCurrentImage“ zugreifen können. In der beim Erreichen des Sprites einmalig ausgeführten Prozedur „beginSprite“ speichern wir den mit dem Sprite verknüpften Darsteller in der Property „pMem“, um diesen Schritt nicht bei jedem Mausklick unnötigerweise wiederholen zu müssen. Beim Loslassen der Maustaste ermittelt die Prozedur „mouseUp“ zuerst die Nummer der angeklickten Textzeile und bricht ab, wenn es sich nicht um eine gültige Auswahl handelt, also etwa um einen Textrahmen. Andernfalls importiert die Prozedur den Text der angeklickten Zeile in die Variable „titel“ und sucht mithilfe der Funktion „getOne()“ aus der globalen Textliste den zugehörigen Dateipfad heraus. Dann wird der Zeiger auf das aktuelle Bild, also auf die Variable „gCurrentImage“ gesetzt und die oben bereits vorgestellte Prozedur „showImage“ aufgerufen, welche die Aktualisierung der Bild- und Textanzeige übernimmt. Im Unterschied zu den Behaviors der Navigationsbuttons übergeben wir hier allerdings den Wert Null beim Aufruf, da kein Sprung zum vorhergehenden oder folgenden Bild erfolgen soll, sondern lediglich eine Aktualisierung der Anzeige anhand des bereits gesetzten Zeigers „gCurrentImage“. Die hier beschriebene Form des Aufrufs verwenden wir übrigens auch in der Prozedur „startMovie“, die Director automatisch beim Start der Wiedergabe ausführt.

13: Prozedur „search“ aufrufen

```

[global]
on mouseUp
  search member("suchtext").text
end

```

14: Eingabetaste erkennen

```

[global]
on beginSprite me
  sprite(me.spriteNumber).member.text = ""
end
on keyDown
  if the key = RETURN then
    search member("suchtext").text
  else
    pass
  end if
end

```

15: Behavior der Suchergebnisliste

```

property pMem
global gTextList, gImageList, gCurrentImage
on beginSprite me
  pMem = sprite(me.spriteNumber).member
  pMem.text = ""
end
on mouseUp me
  ml = pointToLine(sprite me.spriteNumber, the mouseLoc)
  if ml < 1 then exit
  titel = pMem.line[ml]
  memname = getOne(gTextList, titel)
  gCurrentImage = getPos(gImageList, memname)
  showImage 0
end

```

**Der Button „Bild löschen“.** Das Behavior des Buttons „Bild löschen“ (Abbildung 16) ruft die im Filmskript gespeicherte Prozedur „deleteImage“ auf (siehe Abbildung 12). Diese löscht den aktuellen Eintrag in der Bildliste – also den Eintrag, auf den die Variable „gCurrentImage“ zeigt – und entfernt auch den entsprechenden Eintrag in der Textliste, wobei hier aufgrund der möglicherweise verschiedenen Elementfolge zunächst die zu löschende Position ermittelt werden muss – durch Suchen der Position des aktuellen Titels innerhalb der Liste.

Sofern ein Textdarsteller für das zu entfernende Bild angelegt wurde, löschen wir diesen mithilfe des Befehls „erase member“. Anschließend schalten wir auf das vorhergehende Bild um. Sofern auch das letzte Bild entfernt wurde, erzeugen wir wieder den Anfangszustand, indem die Textan-

zeigen gelöscht und als Bild wieder unser Platzhalter angezeigt wird.

**Speichern und Laden des aktuellen Zustands.** Unser Film soll sämtliche Änderungen beim Programmende automatisch speichern und diese beim nächsten Start auch automatisch wieder laden können. Die Sicherung der globalen Listen „gImageList“ und „gTextList“ realisieren wir daher mit der Funktion „setPref“. Diese erwartet als Parameter einen Dateinamen; der Ordner, in dem die angegebene Textdatei erstellt wird, ist fest vorgegeben. Während des Authorings wird der Ordner „Prefs“ innerhalb des Director-Programmordners verwendet (Abbildung 17), im Projektor hingegen ist nur der Ordner „Prefs“ im Ordner des Projektors erreichbar. Wenn Sie „setPref()“ in einem Shockwave-Film verwenden, speichert das Plug-in den Ord-

ner „Prefs“ übrigens im Ordner „Systemordner:Systemerweiterungen:Macromedia:“ und bietet so die einzige Möglichkeit zur Speicherung externer Daten auf dem System des Anwenders, da sich Xtras wie FileIO aus Sicherheitsgründen nicht in Shockwave verwenden lassen.

**Erweiterungen.** Das vorliegende Beispiel bietet an vielen Stellen Raum für eigene Erweiterungen. So kann es eventuell sinnvoll sein, die Möglich-

### Das vorliegende Beispiel bietet an vielen Stellen Raum für eigene Erweiterungen, etwa um mehrere Alben speichern zu können

keit zur getrennten Speicherung mehrerer Alben vorzusehen. Ferner prüft die vorliegende Version nicht, ob Sie im Dialog „Bild laden“ auch wirklich eine Bilddatei ausgewählt oder möglicherweise einen anderen Dateityp geöffnet haben.

Der Einbau der gewohnten und praktischen Funktionalität der Zwischenablage würde die Übernahme von Text aus anderen Anwendungen erleichtern. Wie Sie die Standard-Shortcuts in Director-7-Filmen aktivieren, lesen Sie beispielsweise in dem Beitrag „Director 7-Text über die Zwischenablage kopieren“, den Sie auf der Website [www.director-workshop.de/data/workshop/clipboard\\_lingo.html](http://www.director-workshop.de/data/workshop/clipboard_lingo.html) finden *Gerd Gillmaier* ■

16: LösCHFunktion



17: Dateipfad der Preferences

