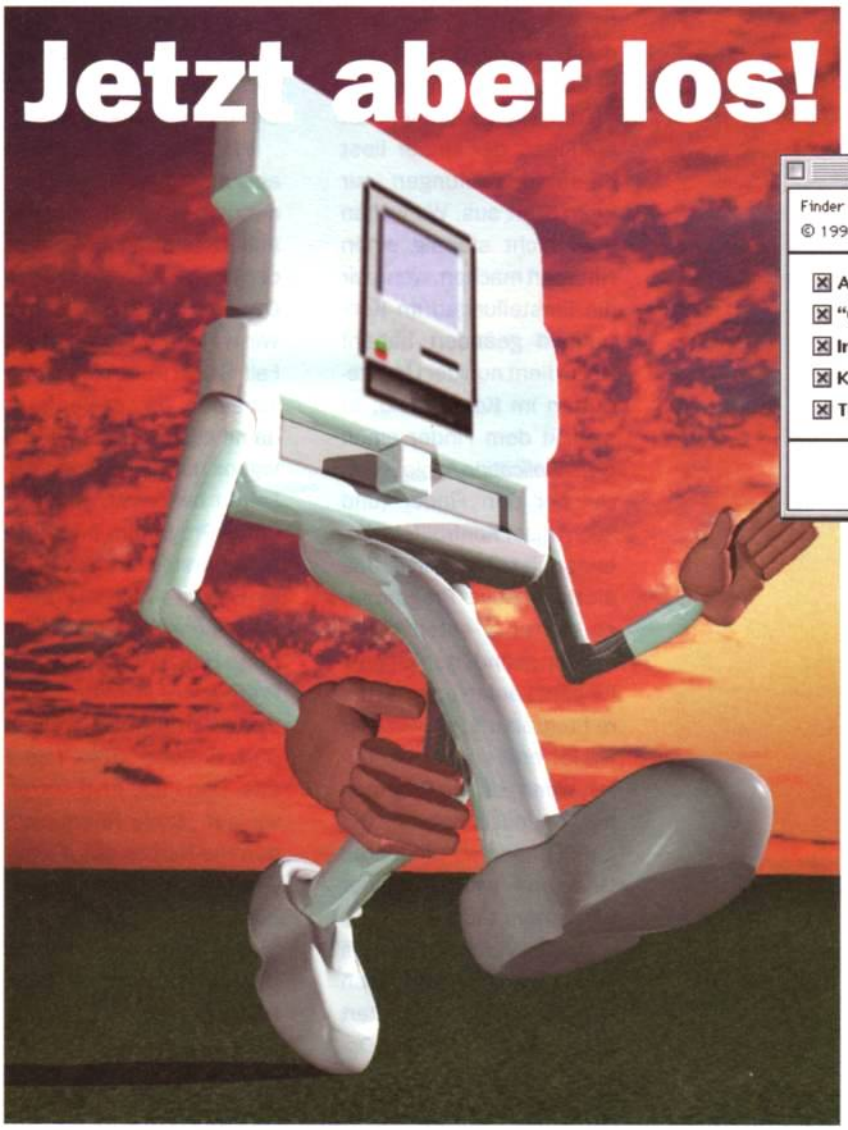

Versteckte Funktionen im Finder aktivieren

Seit System 7 Pro kennt der Finder einige teilweise sehr nützliche Funktionen, welche aber normalerweise abgeschaltet sind, da sie nicht offiziell abgesegnet wurden. Durch die einfache Installation eines Gestalt-Selectors kann man sie jedoch freischalten. Und genau das wollen wir heute mit einer eigenen kleinen Systemerweiterung inklusive dazugehörigem Kontrollfeld programmieren!

Die Vorgabe ist recht einfach: Der Finder fragt beim Start einen Gestalt-Selector vom Typ ‚fnd≈' ab (das letzte Zeichen ist eine Doppeltilde mit einem ASCII-Code $C5, mit Option-X auf der Tastatur zu erreichen). Er erwartet eine Bit-Maske, die einzelne Funktionen freischaltet:

Ist Bit 0 gesetzt, kann man durch Draggen eines Icons bei gedrückter Control-Taste ein Alias erzeugen. Sehr praktisch, um Aliases auf dem Desktop oder im Apfel-Menü-Ordner abzulegen!

Bit 1 fügt dem Finder im Ablage-Menü unterhalb von „Finden..." einen neuen Menüpunkt „Original finden" hinzu. Klickt man im Finder ein Alias an, kann man über diesen Menüpunkt – ohne Umwege über die Informationen-Dialogbox – den Ordner öffnen, in dem das Original zum Alias liegt. Durch das Setzen von Bit 2 kann man mit Command-Backspace alle ausgewählten Dateien in den Papierkorb bewegen.

Bit 3 schließlich schaltet die teilweise sehr langsame Zoom-Rechteck-Animation im Finder ab.

Ab System 7.5.3 gibt es zudem noch den Gestalt-Selector ‚fndμ' (das sonderbare Zeichen ist der griechische Buchstabe mü mit einem ASCII-Code $B5, Option-M), welcher es erlaubt, das transluzente Draggen im Finder auf PowerMacs abzuschalten. Transluzent bedeutet dabei, daß das Icon unter dem Mauscursor als heller Schatten gedraggt wird statt als einfacher Rahmen. Ein Wert von $00 steht dabei für Draggen mit dem Transluzenzeffekt, $0F für das herkömliche Draggen.

Unsere Aufgabe

Wir wollen nun eine kleine Systemerweiterung schreiben, welche obige Gestalt-Selektoren anmeldet und dem Finder unsere Wunschkonfiguration übergibt. Damit der Benutzer seine Konfiguration leicht einstellen kann, liefern wir ein kleines Kontrollfeld mit.

Dabei gibt es zwei Probleme. Das erste ist die Kommunikation unseres Kontrollfeldes mit der Systemerweiterung. Hier bieten sich verschiedene Möglichkeiten an: eine gemeinsame Resource oder aber ein zusätzlicher Gestalt-Selector, mit welchem unsere Systemerweiterung einen Pointer auf die beiden Rückgabewerte übergibt.

Versteckte Funktionen im Finder aktivieren

```
/***
 *  Finder Feature INIT.cp
 *  (c)1996 Maxon
 * Autor: Markus Fritze
 ***/

#include <Gestalt.h>

static void         MyGestaltFuncEnd();

/***
 *  Unsere (sehr kleine)
 *  Gestalt-Selector-Funktion
 ***/
static pascal OSErr MyGestaltFunc(
          OSType selector, long *response)
{
    long    *lp =
      static_cast<long*>(&MyGestaltFuncEnd);

    // Finder Flags
    if(selector == 0x666E64C5L) {
        *response = lp[0];

    // Translucent dragging
    } else if(selector == 0x666E64B5L) {
        *response = lp[1];

    // der eigene Selector
    } else if(selector == 'FPat') {
        *response = static_cast<long>(lp);
    }
    return noErr;
}
// Hiermit markieren wir das Ende der obigen
// Funktion, um die Länge des Codeblockes zu
// errechnen
static void         MyGestaltFuncEnd()
{
}

/***
 * Dieser Code wird beim Neustart aufgerufen
 ***/
void            main(void)
{
    // Die Länge unseres Code-Fragmentes
    Size    theSize =
      static_cast<long>(&MyGestaltFuncEnd) -
      static_cast<long>(&MyGestaltFunc);

    // Speicher im System-Heap allozieren
    // + 2 longs für unsere Variablen
    Ptr     theData = ::NewPtrSysClear(theSize +
                          sizeof(long) * 2);
    // genügend Speicher vorhanden?
    if(!theData) {
        ::SysBeep(1);
    } else {
        // unsere Routine in den System-Heap
        // verschieben
        ::BlockMove(MyGestaltFunc, theData,
                    theSize);

        // Variablen auf Defaults setzen
        long    *thePref =
         static_cast<long*>(theData
                          + theSize);
        thePref[0] = 0x07;
        thePref[1] = 0x00;

        // Voreinstellungen übernehmen
        long    **thePrefH =
         static_cast<long**>(GetResource('Pref',
                                    -4048));
        if(thePrefH) {                          →
```

So kann unser Kontrollfeld dann sehr einfach die Werte ändern.

Das zweite Problem ist kniffliger: der Finder liest diese Einstellungen nur beim Start aus. Wir wollen aber nicht ständig einen Neustart machen, wenn wir die Einstellungen im Kontrollfeld geändert haben! Dazu dient nun der Update-Button im Kontrollfeld: er schickt dem Finder einen *Quit Application*-AppleEvent, welcher den Finder (und auch unser Kontrollfeld) beendet. Ist der Finder das einzige aktive Programm, so startet er sofort neu – mit den geänderten Einstellungen! Laufen noch weitere Programme, so wird der Finder erst wieder gestartet, wenn wir alle Programme beendet haben. Bis dahin haben wir ein leeres Desktop – weil der Finder nicht mehr läuft.

Dieses Verhalten können wir auch nutzen: zum einen können wir beim Neustart des Finders Command-Option gedrückt halten, um die Desktop-Datenbank neu aufzubauen. Auch hier ist somit kein Neustart des Rechners nötig! Zum anderen können wir zusätzlichen Speicher gewinnen, indem wir den Finder beenden, nachdem wir unser Wunschprogramm gestartet haben. Dies bringt uns in der Regel aber nur wenig Nutzen, da die meisten Programme nachträglich hinzugekommenen Speicher nicht nutzen können.

Die Systemerweiterung

Eine Systemerweiterung hat normalerweise den Dateityp *INIT* und sie befindet sich im „Systemerweiterungen"-Ordner innerhalb des Systemordners. Das MacOS führt dann alle Resourcen vom Typ *INIT* innerhalb dieses Programms aus.

INIT-Resourcen können aber auch in Kontrollfeldern mit dem Dateityp *cdev* enthalten sein. Auch sie werden beim Neustart des MacOS automatisch ausgeführt. Wir werden uns in diesem Fall für ein Kontrollfeld entscheiden, da der Benutzer damit auch eine kleine Dialogbox mit Einstellmöglichkeiten bekommt, wenn er unser Programm öffnet.

Als erstes erstellen wir ein 68k-C++-MacOS-Projekt für unsere Systemerweiterung. Hierzu öffnen wir die Preferences im Metrowerks CodeWarrior und wählen dort Project: 68k-Project aus. Den Projekttyp setzen wir auf „Code Resource", den File-Namen auf „Finder Feature INIT.rsrc", Creator auf ‚RSED', Filetyp auf ‚rsrc' (damit ist unsere erzeugte Datei ein ResEdit-File). Das ResType setzen wir auf *INIT*, die ResID auf –4048.

Beim Compilieren wird somit eine ResEdit-Datei „Finder Feature INIT.rsrc" erzeugt, welche eine einzelne *INIT*-Resource mit unserem Programmcode enthält. Diese Resource werden wir später – in einem zweiten Schritt – in unser Kontrollfeld mergen.

Eine Systemerweiterung wird ohne zusätzliche Parameter beim Neustart des Mac aufgerufen. Da die *INIT*-Resource automatisch wieder freigegeben wird, müssen wir unseren residenten Programmcode selbst auf den System-Heap kopieren. Dazu gibt es zwei Möglichkeiten:

Wir erstellen eine extra Code-Resource, welche wir mit den Attributen *Preload* und *System* versehen. Schon wird das Teil beim

Ausführen unserer Systemerweiterung automatisch in den System-Heap geladen. Wir müssen das Resource-Handle nur noch mit *DetachResource()* abmelden, damit es beim Verlassen von *main()* nicht wieder freigegeben wird. Nachteil des Verfahrens ist jedoch, daß wir ein drittes Projekt-File brauchen.

Wir allozieren deswegen selbst Speicher im System-Heap und verschieben unseren Programmcode einfach mit *BlockMove()* dorthin. Ich habe eine Dummy-Funktion hinter den Programmcode für die Gestalt-Funktion gelegt, um die Länge der Gestalt-Funktion zu errechnen. Früher habe ich einfach immer die Funktion direkt hinter der kopierten Funktion genommen, aber als ich einmal die Reihenfolge der Funktionen verändert hatte, stürzte die Systemerweiterung sofort ab. Deswegen gibt es bei mir nun eine extra Dummy-Funktion, da denkt man dann sofort an die Reihenfolge ... Nun lesen wir noch unsere Voreinstellungen ein und kopieren sie ebenfalls in den Speicherblock im System-Heap.

Als letztes melden wir unsere drei Gestalt-Funktionen mit *NewGestalt()* an: zwei für den Finder, eine für die Kommunikation mit unserem Kontrollfeld. Da die Gestalt-Funktion den Selectorcode übergeben bekommt, reicht eine einzelne Funktion für alle drei Selectoren.

Die Gestalt-Funktion ist dann trivial: für die Finder-Selectoren werden einfach unsere Werte zurückgegeben, für unseren eigenen Selector ein Zeiger auf diese beiden Werte. Fertig!

Das Kontrollfeld

Ein Kontrollfeld hat den Dateityp *cdev* und enthält eine *cdev*-Resource. Zieht man ein Kontrollfeld auf den Systemordner, wird es automatisch in den Kontrollfelder-Ordner kopiert. Kontrollfelder dürfen auch Systemerweiterungen enthalten.

Wir erstellen nun unser zweites 68k-C++-MacOS-Projekt. Zu diesem Projekt fügen wir den Source-Code „Finder Feature Enabler.cp", die Rez-Datei „Finder Feature Enabler.r" und unsere „Finder Feature INIT.rsrc", welche die Systemerweiterung enthält. Für das Übersetzen der Rez-Datei brauchen wir einen CodeWarrior 8.

Nun stellen wir auch hier in den Preferences den Projekttyp auf „Code Resource", den File-Namen auf „Finder Feature Enabler", Creator auf ‚FPat', Type auf ‚cdev'. Der ResType ist ebenfalls ‚cdev' und die ResId ist –4064.

Wenn wir unser Projekt nun mit Make durchlaufen lassen, wir zum einen unser C++-Kontrollfeld übersetzt und daraus eine *cdev*-Resource erzeugt, zum anderen werden die Resourcen mit dem Rez-Compiler übersetzt und unsere *INIT*-Resource wird ebenfalls übertragen. Kurzum: wir erzeugen unser fertiges Kontrollfeld!

Die Routinen *GetGestalt()*, *GetItemHandle()*, *GetCheck()* und *SetCheck()* sollten trivial sein. Näheres kann man im Sourcecode sehen.

Die Routine *FindAProcess()* sucht einen laufenden Prozeß anhand seines Creators und Filetypes und gibt die Prozeßseriennummer zurück. Wir benutzen

```
77:             thePref[0] = (*thePrefH)[0];
78:             thePref[1] = (*thePrefH)[1];
79:             ReleaseResource(
80:              static_cast<Handle>(thePrefH));
81:         }
82:
83:         // und als Gestalt-Funktion anmelden
84:         ::NewGestalt(0x666E64C5L,
85:               SelectorFunctionProcPtr(theData));
86:         ::NewGestalt(0x666E64B5L,
87:               SelectorFunctionProcPtr(theData));
88:         ::NewGestalt('FPat',
89:               SelectorFunctionProcPtr(theData));
90:     }
91: }
```

```
 1: /***
 2:  *  Finder Feature Enabler.cp
 3:  *  (c)1996 Maxon
 4:  * Autor: Markus Fritze
 5:  ***/
 6:
 7: #include <Gestalt.h>
 8:
 9: /***
10:  *  einen Gestalt-Selector abfragen
11:  ***/
12: static long     GetGestalt(OSType iSelector)
13: {
14: long    theResponse;
15:
16:   if(::Gestalt(iSelector, &theResponse) != noErr)
17:     theResponse = 0L;
18:   return theResponse;
19: }
20:
21: /***
22:  *  ControlHandle eines Items in einer Dialogbox
23:  *  ermitteln
24:  ***/
25: static ControlHandle    GetItemHandle(
26:     const DialogPtr iDialog, const short iItemNo)
27: {
28: short   theItemType;
29: Handle  theItem;
30: Rect    theBox;
31:
32:   ::GetDialogItem(iDialog, iItemNo, &theItemType,
33:                   &theItem, &theBox);
34:   return ControlHandle(theItem);
35: }
36:
37: /***
38:  *  Checkbox abfragen
39:  ***/
40: static Boolean  GetCheck(const DialogPtr iDialog,
41:                          short iItemNo)
42: {
43:   return ::GetControlValue(
44:          GetItemHandle(iDialog, iItemNo)) != 0;
45: }
46:
47: /***
48:  *  Checkbox setzen
49:  ***/
50: static  void    SetCheck(const DialogPtr iDialog,
51:                    short iItemNo, Boolean iCheck)
52: {
53:   ::SetControlValue(GetItemHandle(iDialog,
54:                             iItemNo), iCheck);
55: }
56:
```

diese Routine um den laufenden Finder zu suchen. Man kann sie natürlich auch allgemein nutzen.

Mit *AEQuitFinder()* schikken wir dem Finder den Quit Application AppleEvent. Der Finder beendet sich (und auch alle Kontrollfelder) und startet sofort neu, wenn er das einzige laufende Programm im Speicher war. Ansonsten haben wir vorerst ein leeres Desktop, bis wir alle anderen Programme beendet haben.

Der eigentliche Programmcode für das Kontrollfeld steckt in *main()*. Ein Kontrollfeld ist quasi ein kleines Programm ohne eigenen Event-Dispatcher und in der Regel auch ohne eigene Menüs. Statt dessen besteht es nur aus einer einzigen nichtmodalen Dialogbox. Der Finder übernimmt fast alle Standardaktionen für solche Dialogboxen, so daß wir uns voll auf die Funktionalität konzentrieren können.

Der wichtigste Parameter der *main()*-Funktion ist die Message. Ein Kontrollfeld erhält viele verschiedene Messages. Wir brauchen in unserem Beispiel jedoch nur einige wenige:

Messages an ein Kontrollfeld

Die Message „macDev" erhält man nur, wenn eine mach-Resource –4064 mit dem Inhalt 0x0000FFFF existiert – diese steht in unserem Rez-File. Hier überprüft das Kontrollfeld, ob es auf diesem Mac überhaupt funktioniert. Als Rückgabe wird einfach true oder false zurückgegeben. Bei uns überprüfen wir einfach, ob ein System 7.1.2 oder neuer vorhanden ist.

Die Message „initDev" ermöglicht es uns, die Checkboxen auf die richtigen Vorgabewerte zu setzen, bevor die Dialogbox des Kontrollfeldes erscheint. Wir fragen unsere Gestalt-Selektoren ab und setzen unsere Checkboxen. Transluzentes Draggen steht erst ab System 7.5.3 zur Verfügung, deswegen disablen wir die Checkbox, wenn ein älteres System vorhanden ist.

Den Update-Button disablen wir nur, wenn die Systemerweiterung beim Start des MacOS nicht geladen wurde. Dies erkennen wir daran, daß unser *FPat*-Selector installiert ist.

Die Message „hitDev" erhalten wir, wenn der Benutzer ein Item in der Dialogbox anklickt. Hier toggeln wir die Checkboxen, ändern die Werte in den Voreinstellungen und in der Systemerweiterung. Klickt der User auf den Update-Button, so starten wir den Finder nach einer Sicherheitsabfrage neu.

Damit wäre unser Kontrollfeld schon fertig. Trotzdem noch einige Worte zu den weiteren Messages.

„closeDev" wird dem Kontrollfeld geschickt, wenn das Kontrollfeld geschlossen wird. Falls wir bei „initDev" Speicher alloziert haben, so müssen wir ihn hier wieder freigeben! Bei „initDev" geben wir das allozierte Speicher-Handle als Rückgabecode der *main()*-Funktion zurück. Wir bekommen es dann als Parameter bei den Messages übergeben. Allozieren wir – wie in unserem Beispiel – keinen Speicher, geben wir „cdevUnset" zurück. Weitere Fehlermeldungen sind „cdevGenErr" – für einen allgemeinen Fehler; „cdevMemErr" – falls zuwenig Speicher vorhan-

```
57: /***
58:  *  Eine Applikation in der Prozessliste nach
59:  *  ihrem Filetype und Creator suchen.
60:  ***/
61: static OSErr     FindAProcess(OSType iType,
62:                                 OSType iCreator,
63:                 ProcessSerialNumberPtr oProcessSN)
64: {
65: ProcessInfoRec  theProcessInfo;
66: FSSpec          theProcessFSSpec;
67: Str31           theProcessName;
68: OSErr           theErr = noErr;
69:
70:   // start at the beginning of the process list
71:   oProcessSN->lowLongOfPSN = kNoProcess;
72:   oProcessSN->highLongOfPSN = kNoProcess;
73:
74:   // initialize the process information record
75:   theProcessInfo.processInfoLength =
76:                             sizeof(ProcessInfoRec);
77:   theProcessInfo.processName =
78:                         (StringPtr)&theProcessName;
79:   theProcessInfo.processAppSpec =
80:                                 &theProcessFSSpec;
81:
82:   while((
83:    theProcessInfo.processSignature != iCreator ||
84:           theProcessInfo.processType != iType) ||
85:             theErr != noErr)
86:   {
87:     theErr = GetNextProcess(oProcessSN);
88:     if(theErr == noErr)
89:       GetProcessInformation(oProcessSN,
90:                             &theProcessInfo);
91:   }
92:   return theErr;
93: }
94:
95: /***
96:  *  Unsere Konstanten für das Senden von
97:  *  AppleEvents an den Finder. Siehe auch
98:  *  Kapitel 8 des Apple Event Registry
99:  ***/
100: const OSType    kFinderSig = 'FNDR';
101: const OSType    kSystemType = 'MACS';
102:
103: /***
104:  *  Den Finder per AppleEvent beenden
105:  ***/
106: static OSErr    AEQuitFinder()
107: {
108: // Der erstellte Event
109: AppleEvent    theEvent;
110:
111: // Descriptor der Prozessnummer des Finders
112: AEDesc        theAddressDesc;
113: ProcessSerialNumber theProcess;
114:
115: OSErr         theErr;
116:
117:   // Prozessnummer des Finders ermitteln
118:   if(FindAProcess(kFinderSig, kSystemType,
119:                   &theProcess))
120:     return procNotFound;
121:
122:   // Seriennummer in einen Descriptor packen
123:   // um ein Ziel für den Apple Event zu
124:   // definieren
125:   theErr = AECreateDesc(typeProcessSerialNumber,
126:                         (Ptr)&theProcess,
127:                         sizeof(theProcess),
128:                         &theAddressDesc);
129:   if(theErr) return theErr;
130:
```

→

den ist und „cdevResErr", falls eine Resource fehlt.

„nulDev", „updateDev", „activDev", „deactivDev" und „keyEvtDev" entsprechen den normalen Mac-Events. Wir brauchen sie üblicherweise nur, wenn wir Useritems in unser Dialogbox haben, die wir selbst zeichnen und updaten müssen, oder wenn wir Tastaturshortcuts unterstützen wollen.

„undoDev", „cutDev", „copyDev", „pasteDev" und „clearDev" entsprechen den Menüpunkten im Bearbeiten-Menü. Wir brauchen sie wahrscheinlich nur für Textedit-Felder in einer Kontrollfeld-Dialogbox.

Als letzter Event ist der „cursorDev" definiert worden, mit dem man den Mauszeiger innerhalb der Dialogbox einfach ändern kann.

Die Resourcen

Neben der *INIT*- und der *cdev*-Resource brauchen wir noch einige weitere Resourcen für unser fertiges Kontrollfeld:

Ein Kontrollfeld besteht aus einer *DITL*-Resource und der dazugehörigen *nrct*-Resource, welche Rechtekke beschreibt, die Gruppen innerhalb der *DITL*-Resource zusammenfassen. Die *nrct*-Resource stammt noch aus System-6-Zeiten, wird aber auch heute noch benutzt.

Ferner definieren wir eine *mach*-Resource, welche dafür sorgt, daß wir eine „macDev"-Message bekommen. Hier entscheiden wir, ob das Kontrollfeld auf dem aktuellen Macintosh läuft.

Die *BNDL*-, *FREF*-, *FPat*-, *ICN#*-, *icl4*- und *icl8*-Resourcen sorgen dafür, daß unser Kontrollfeld auch ein eigenes Icon bekommt. Beim Systemstart zeigen viele Systemerweiterungen das Programm-Icon an. Den dafür nötigen Programmcode habe ich hier aus Platzgründen gespart. Man kann ihn aber in diversen Mailboxen finden (Stichwort: *ShowInitIcon*).

Zu guter Letzt noch eine *vers*-Resource um die Sache abzurunden...

Debugging

Systemerweiterungen lassen sich leider sehr schlecht debuggen, denn Highlevel-Debugger, wie der Metrowerks Debugger, funktionieren bei Systemerweiterungen noch nicht. Hier muß man auf einen Lowlevel-Debugger wie MacsBug zurückgreifen oder aber auf Steve Jasiks „The Debugger", der sogar als Sourcecode-Debugger bei Systemerweiterungen funktioniert! Leider ist er sehr teuer. Trotzdem ist er für den professionellen Entwickler empfehlenswert.

Kontrollfelder lassen sich sehr einfach mit der *cdev*-Shell debuggen, die man ebenfalls in diversen Mailboxen finden kann.

So, damit bin ich bei unserem Exkurs zum Thema Kontrollfelder am Ende angelangt. Ich hoffe, es war mal wieder etwas Interessantes dabei – zumal gerade das Kontrollfeld sehr interessante Funktionen bietet.

MFR

Literatur:
New Inside Macintosh, More Macintosh Toolbox (ISBN 0-201-63299-3, $34.95).
Kapitel 8: Control Panels
The Debugger V2 & MacNosy ($256, Internet: macnosy@jasik.com)

```
131:   // Apple Event erstellen
132:   theErr = AECreateAppleEvent(kCoreEventClass,
133:                               kAEQuitApplication,
134:                               &theAddressDesc,
135:                            kAutoGenerateReturnID,
136:                               kAnyTransactionID,
137:                               &theEvent);
138:   if(theErr) return theErr;
139:
140:   // und abschicken...
141:   theErr = AESend(&theEvent, nil, kAENoReply |
142:           kAEAlwaysInteract | kAECanSwitchLayer,
143:           kAENormalPriority, kAEDefaultTimeout,
144:           nil, nil);
145:
146:   // hier kommen wir wohl nie an, da der Finder
147:   // auch alle Kontrollfelder beendet und somit
148:   // auch uns. Falls es jedoch einen Fehler gibt
149:   // laden wir sehr wohl hier!
150:   AEDisposeDesc(&theEvent);
151:   return theErr;
152: }
153:
154: /***
155:  * unser cdev-Funktionsdispatcher
156:  ***/
157: pascal long      main(short iMessage, short iItem,
158:                   short iNumItems, short iPrivate,
159:                         const EventRecord *iEvent,
160:                               long iStorageValue,
161:                                DialogPtr iDialog)
162: {
163: #pragma unused (iNumItems, iPrivate)
164: #pragma unused (iEvent, iStorageValue)
165:
166:   switch(iMessage)
167:   {
168:   // Test, ob das Kontrollfeld auf diesem Rechner
169:   // läuft:
170:   case macDev:
171:
172:   // wir setzen einfach mal System 7.1.2 voraus
173:     return GetGestalt(gestaltSystemVersion) >=
174:                                           0x712;
175:
176:
177:   // Initialisierung
178:   case initDev:
179:     long    theVal1 = GetGestalt(0x666E64C5L);
180:     SetCheck(iDialog, 4,
181:              (theVal1 & 0x01) == 0x01);
182:     SetCheck(iDialog, 5,
183:              (theVal1 & 0x02) == 0x02);
184:     SetCheck(iDialog, 6,
185:              (theVal1 & 0x04) == 0x04);
186:     SetCheck(iDialog, 7,
187:              (theVal1 & 0x08) == 0x08);
188:
189:     if(GetGestalt(gestaltSystemVersion) >= 0x753)
190:     {
191:       SetCheck(iDialog, 8,
192:        (GetGestalt(0x666E64B5L) & 0x01) != 0x01);
193:     } else {
194:       ::HiliteControl(GetItemHandle(iDialog, 8),
195:                       kControlDisabledPart);
196:     }
197:     // INIT nicht installiert?
198:     if(GetGestalt('FPat') == 0L)
199:     {
200:       // dann den Update Button disablen
201:       ::HiliteControl(GetItemHandle(iDialog, 3),
202:                       kControlDisabledPart);
203:     }
```

→

```
204:      break;
205:
206:    // Ein Item wurde angeklickt
207:    case hitDev:
208:      if(iItem >= 4 && iItem <= 8) // Checkboxes
209:      {
210:        // Checkbox togglen
211:        SetCheck(iDialog, iItem,
212:                 !GetCheck(iDialog, iItem));
213:
214:        long *thePref = static_cast<long*>(
215:                          GetGestalt('FPat'));
216:        if(thePref)
217:        {
218:          if(iItem < 8)
219:          {
220:            // Bitmaske mit den Flags
221:            thePref[0] &= ~(1 << (iItem - 4));
222:            if(GetCheck(iDialog, iItem))
223:              thePref[0] |= 1 << (iItem - 4);
224:
225:          } else {
226:            // Translucent dragging aus
227:            thePref[1] = 0x0FL;
228:
229:            if(GetCheck(iDialog, 8))
230:              thePref[1] = 0x00L; // an
231:          }
232:        }
233:
234:        // Voreinstellungen in der Resource updaten
235:        long    **thePrefH =
236:                  static_cast<long**>(
237:                   GetResource('Pref', -4048));
238:        if(thePrefH)
239:        {
240:          (*thePrefH)[0] = thePref[0];
241:          (*thePrefH)[1] = thePref[1];
242:          Handle theH =
243:                   static_cast<Handle>(thePrefH);
244:          ChangedResource(theH);
245:          WriteResource(theH);
246:          ReleaseResource(theH);
247:        }
248:
249:      } else if(iItem == 3) // Update
250:      {
251:        // noch eine Sicherheitsabfrage
252:        if(StopAlert(-4048, nil) == 1)
253:          AEQuitFinder();
254:      }
255:      break;
256:
257:    case closeDev:      // Kontrollfeld schließen
258:    case nulDev:        // Null-Event
259:    case updateDev:     // Update-Event
260:    case activDev:      // Activate-event
261:    case deactivDev:    // Deactivateevent
262:    case keyEvtDev:     // Key down/auto-Event
263:    case cursorDev:     // Mauscursor setzen
264:    // Ein Menüpunkt aus dem Bearbeiten Menü wurde
265:    // gewählt:
266:    case undoDev:
267:    case cutDev:
268:    case copyDev:
269:    case pasteDev:
270:    case clearDev:
271:      break;
272:    }
273:
274:    // Wir haben keine Daten alloziert
275:    return cdevUnset;
276: }
```