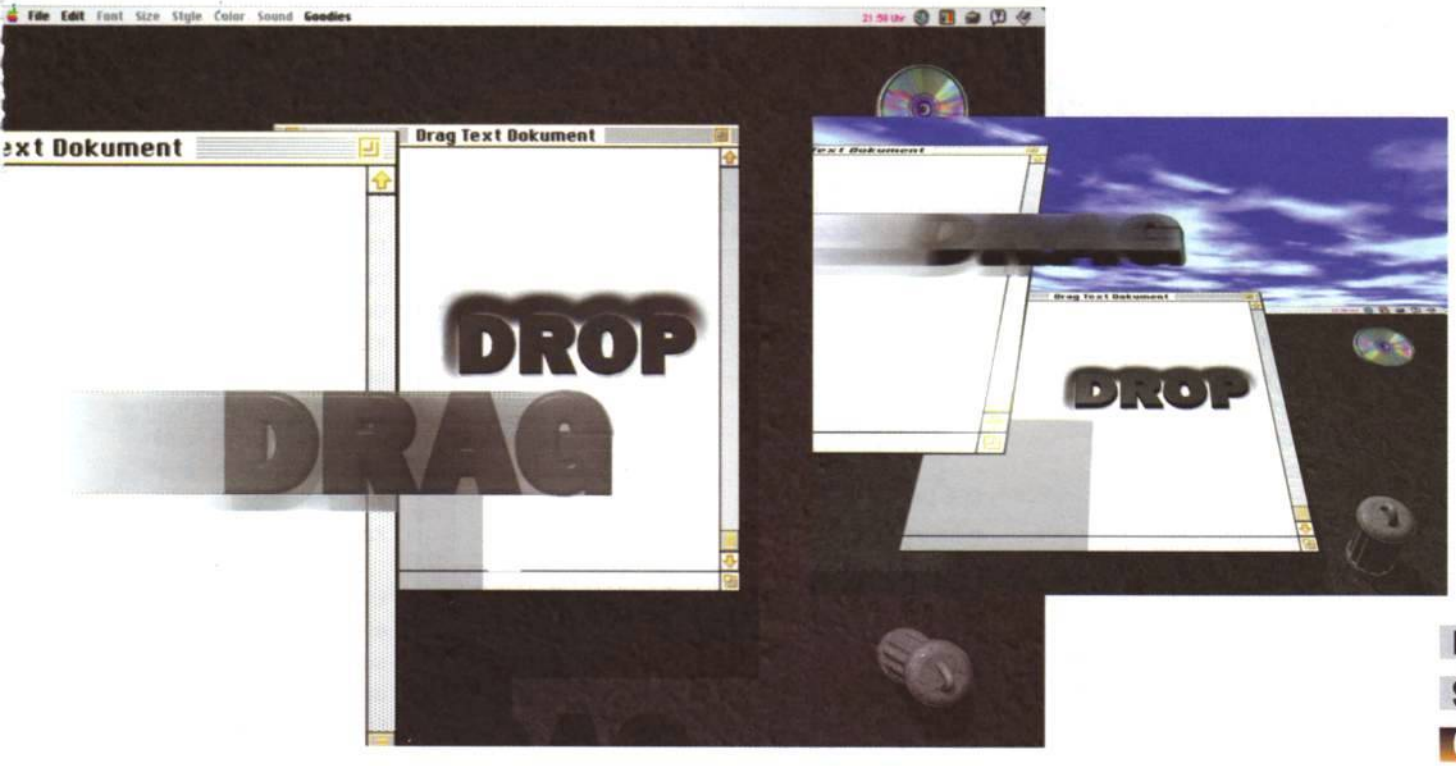


Drag & Drop-Manager

Viele neue Technologien hat Apple in den letzten 2 Jahren vorgestellt, aber nur wenige waren so praktisch wie die Implementation des Drag & Drop-Managers. Gerade, weil das MacOS dafür bekannt ist, daß es sehr anwenderfreundlich ist, drängt sich einem die Frage auf: Warum erst jetzt?

Im Finder nutzt man es täglich: man greift mit der Maus ein oder mehrere Icons und schiebt es/sie in einen Ordner oder auf ein anderes Laufwerk. Schiebt man ein Dokument auf ein Programm, so wird das Programm dieses Dokument öffnen. Auch innerhalb von Programmen, wie Quark XPress oder EMAGICs Logic, kann man Objekte von einem Fenster in ein anderes verschieben. Warum aber kann man nicht einfach ein File vom Finder in ein Fenster eines bereits laufenden Programmes schieben, um z.B. eine Grafik in einen Text einer Textverarbeitung einzufügen? Oder warum kann man nicht einen Textabschnitt aus MacWrite einfach in den Papierkorb verschieben? Genau dies ermöglicht nun der Drag & Drop-Manager!

Systemvoraussetzungen

Der Drag & Drop-Manager ist ab System 7.5 fest integriert, aber man kann ihn auch in System 7.1 installieren. Die nötigen Systemerweiterungen kann man auf Apples FTP-Server finden. Dazu gehört:

Macintosh Drag and Drop

Diese Systemerweiterung enthält die Funktionalität des Drag & Drop-Managers. Sie ist nur für System 7.1 nötig.

Clipping Extension

Diese Erweiterung erlaubt es dem Finder, Clippings auf dem Desktop abzulegen. Clippings sind Daten-Files, welche beim Drag & Drop aus einem Anwenderprogramm entstehen, wenn man die Daten z.B. auf dem Desktop ablegt. Dies können ein PICT oder Textdaten sein. Standardtypen kann der Finder direkt öffnen und anzeigen. Man kann Clippings direkt in Programme hineindraggen und somit einfügen. Eine sehr komfortable Art eines Albums also. Clippings können übrigens auch direkt in den Papierkorb gezogen werden. So wird ein Objekt zwar gelöscht, man kann es aber problemlos wieder aus dem Papierkorb ziehen!

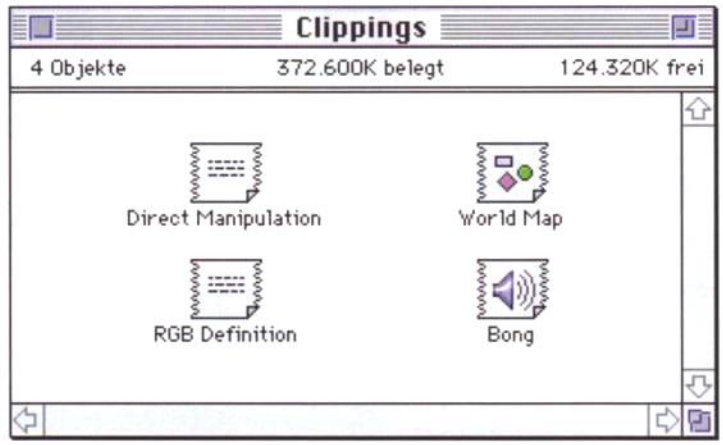
Dragging Enabler

Erlaubt es System 7.1, Drag & Drop zu unterstützen. Ab System 7 Pro bzw. System 7.5 unnötig.

Was ist Drag & Drop?

Verdeutlichen wir uns zuerst einmal, wie eine Drag & Drop-Operation im einzelnen aussieht:

Drag & Drop-Manager



Clippings sind Daten-Files, welche beim Drag & Drop aus einem Anwenderprogramm entstehen, wenn man die Daten z.B. auf dem Desktop ablegt.

Wir wählen mehrere Objekte aus. Klicken wir nun in ein Objekt und halten die Maustaste gedrückt, beginnen wir beim Bewegen der Maus die Umrahmung aller ausgewählten Objekte zu verschieben. Jedes Objekt kann dabei einen anderen Typ haben. So gibt es im Finder z.B. Dateien verschiedenen Typs, Ordner und Laufwerke.

Wir bewegen nun die Maus in ein anderes Fenster des Finders. Im Fensterinnenbereich erscheint ein blauer Rahmen als Zeichen dafür, daß wir hier die Maustaste loslassen dürfen, um unsere Objekte hier abzulegen. Dieses sinnvolle Feedback beim Verschieben ist im Finder übrigens auch erst mit System 7 Pro implementiert worden. Erscheint kein blauer Rahmen, so ist dies ein Zeichen dafür, daß das Fenster mit den gedraggten Objekten nichts anfangen kann. Im Finder könnte dies der Fall sein, wenn man versuchte, Dateien auf ein schreibgeschütztes Medium zu ziehen. Läßt man in einem solchen Fall die Maustaste los, so springen die Icons zurück zu ihrem Ausgangspunkt, und der Drag wird abgebrochen.

Ist der Drag erlaubt, können wir die Maustaste loslassen. Im Finder würde jetzt der Kopiervorgang gestartet werden. In eigenen Programmen sollten wir beim Loslassen noch die Auswahl-Taste abfragen. Ist sie gedrückt, werden unsere Daten nicht verschoben, sondern kopiert. Dies gilt allerdings nur innerhalb unseres eigenen Fensters!

Was für Daten enthält ein Drag & Drop-Vorgang?

Beim Draggen kann man natürlich nicht nur einen Textblock verschieben, man kann - wie der Finder auch – mehrere Objekte gleichzeitig verschieben. Diese Objekte brauchen dabei nicht einmal den gleichen Typ zu haben. Der Finder z.B. erlaubt natürlich das gemeinsame Verschieben von Dateien und Ordnern.

Will man einen Drag starten, so muß man dem Drag & Drop-Manager eine Liste von Objekten übergeben, die alle einen oder mehrere individuelle Flavors, eigene Daten sowie eigene Attribute haben.

Was sind Flavors?

Der Drag & Drop-Manager ist sehr flexibel und effizient ausgelegt. Er erlaubt nicht nur das Verschieben von Textblöcken oder PICT-Daten, sondern kann beliebige Datentypen verschieben. Diese Datentypen nennt der Drag & Drop-Manager „Flavors".

Der bekannteste und wohl auch am meisten genutzte Flavor ist der *HFS-Flavor*, welche einer Datei beschreibt. Er wird im Finder aber auch bei Pakkern genutzt, welche mit Dateien arbeiten. Es gibt ferner noch den *Promise-HFSFlavor*, welcher eine Datei beschreibt, die allerdings erst auf Wunsch erzeugt wird – Näheres dazu folgt weiter unten.

Nehmen wir mal ein Beispiel: Wir haben eine ganz tolle neue Textverarbeitung geschrieben und wollen nun, daß Textblöcke auch verschoben werden können. Würden wir Textblöcke nur innerhalb unseres eigenen Programms verschieben, so reichte es wahrscheinlich sich für einen Textblock den Start-Offset, die Länge und das Dokument zu merken. Ließe der User die Maustaste los, so könnten wir auf Grund dieser drei Informationen den Block verschieben oder kopieren. Da wir aber beim Beginn eines Drags nicht wissen, ob der Anwender den Textblock nicht eventuell doch in ein anderes Programm verschieben will, müßten wir einen Textblock normalerweise immer durch seinen gesamten Inhalt beschreiben. Dies ist natürlich wenig effizient, vor allem wenn man daran denkt, daß so ein Block natürlich auch ziemlich groß werden kann.

Deswegen bietet der Drag & Drop-Manager eine sehr elegante Lösung für unser Problem an: man kann ein Objekt mit mehreren Flavors anbieten! Unsere Textverarbeitung kann einen Textblock also als *TEXT*, *styl* (Text mit Attributen), *PICT* und *RTF* anbieten sowie in einem internen Format. Dabei kann man sicherstellen, daß unser internes Format nur unserem eigenen Programm vom Drag & Drop-Manager angeboten wird. Der Empfänger kann dann entscheiden, welches Format ihm genehm ist.

Da das Erzeugen von allen möglichen Daten nicht nur Zeit, sondern auch viel Speicher kostet, bietet der Drag & Drop-Manager an, diese Flavors nur auf Rückfrage anzubieten. Man kann also sagen: „Klar, Du kannst diese Daten auch als PICT bekommen". Aber erst, wenn der Empfänger sich wirklich für das PICT-Format entscheidet, dann erzeugt man ein PICT aus den Daten! Dazu melden wir einfach beim Drag eine *SendData*-Funktion an, die automatisch aufgerufen wird, wenn Flavors abgefragt werden.

Ferner kann der Absender den Translation-Manager benutzen, um Daten in andere Formate zu übersetzen (z.B. TEXT in RTF zu wandeln). Falls der Absender die Daten erst in ein bestimmtes Format wandeln muß, so sollte er dies ebenfalls im Flavor markieren. Damit hilft er dem Empfänger sich, für einen optimalen Datentyp zu entscheiden. Schließlich wäre es wenig sinnvoll, wenn der Empfänger RTF wählen würde, weil es mehr Informationen als TEXT enthalten kann, wenn der Absender RTF eh nur aus TEXT erzeugt und somit die zusätzlichen Möglichkeiten gar nicht ausnutzt. Dann kostet die Konvertierung nach RTF und

zurück einfach nur Zeit und bringt dem Anwender keine Vorteile.

Drag-Handler

Springen wir also einfach mal ins kalte Wasser ... Wir können unser bestehendes Programm recht angenehm Schritt für Schritt mit dem Support für Drag & Drop ausstatten. Im ersten Schritt werden wir Tracking- und Receive-Routinen für unsere Fenster anmelden. Dann können wir schon einmal Drags empfangen. Das Gerüst kann man dem folgenden Beispielcode entnehmen.

Fangen wir mit dem einfachen Fall an: Um einen Drop zu empfangen, müssen wir einen Receive-Handler beim Drag & Drop-Manager anmelden. Die Handler sind immer fest mit einem Window verbunden – wir müssen den *WindowPtr* beim Anmelden gleich mit übergeben. Dieser Handler wird nun aufgerufen, wenn der Anwender passende Daten in unserem Fenster abgelegt hat. Wir können hier die Daten durchgehen und unserem Fenster hinzufügen.

Ferner müssen wir noch einen Tracking-Handler anmelden. Dieser Handler wird während des Drag-Vorganges aufgerufen, wenn der Anwender mit der Maus unser Fenster erreicht. Hier entscheiden wir, ob wir den Drop akzeptieren wollen. Wenn dies der Fall ist, müssen wir unser Fenster entsprechend hervorheben – mit oben bereits erwähntem blauen Rahmen. Falls wir den Drop nicht akzeptieren, wird unser Receive-Handler auch dann nicht aufgerufen, wenn der Anwender die Maustaste oberhalb unseres Fensters losläßt.

Der Tracking-Handler bekommt eine Message übergeben, die folgende Werte annehmen kann:

dragTrackingEnterHandler

Der Anwender hat mit der Maus zum ersten Mal ein Fenster erreicht, welches zu unserem Tracking-Handler gehört. Hier kann man einige Variablen initialisieren, die unser Handler braucht, z.B. testen, ob wir diesen Drag akzeptieren können. Da wir einen Tracking-Handler für mehrere Fenster gleichen Typs nutzen können, brauchen wir so nur einmal zu testen, ob wir den Drag akzeptieren können.

dragTrackingEnterWindow

Immer wenn die Maus ein Fenster erreicht, für welches unser Tracking-Handler zuständig ist, bekommen wir diese Message. Wir erhalten diese Message also bei jedem unserer Fenster erneut, auch wenn der Anwender die Maus direkt von Fenster A in Fenster B bewegt. Hier könnten wir unser Fenster mit einem blauen Rahmen markieren als Zeichen dafür, daß wir den Drag akzeptieren.

dragTrackingInWindow

Wenn wir Dragging auch von Daten innerhalb unseres Fensters erlauben, müssen wir hier das Highlighten erledigen. Beispiel: der Anwender hat einen Textblock markiert und verschiebt ihn jetzt innerhalb des Fensters. Bei dieser Message rechnen wir die Mausposition im Fenster in eine Einfüge-Cursor-Position um. Auch Online-Scrolling wird hier erledigt. Drag & Drop bei einem Texteditor ist einer der kompliziertesten Fälle von Drag & Drop. Apple liefert immerhin einen Beispielcode für diesen Fall – wenn auch noch mit obligatorischen Fehlern ...

dragTrackingLeaveWindow

Unser Fenster wurde verlassen. Hier müssen wir unsere Markierung des Fensters zurücknehmen.

dragTrackingLeaveHandler

Unser Tracking-Handler wird vorerst nicht mehr aufgerufen, da ein anderes Fenster mit einem anderen Tracking-Handler erreicht oder der Drag abgebrochen wurde. Hier können wir bei dragTrackingEnterHandler allozierten Speicher freigeben.

Der nächste Schritt dürfte das Draggen innerhalb des eigenen Programms und zu anderen Programmen sein. Innerhalb des eigenen Programms ist es recht einfach, denn wir können Referenzen durch die Gegend schieben. Bei Drags in andere Programme müssen wir unsere Referenzen in sinnvolle Datentypen wandeln.

Real Life

Klingt alles recht einfach? Ist es im Prinzip auch, aber im Detail ist gerade das Draggen ins eigene Programm kniffelig, wenn man will, daß die Daten an einer bestimmten Stelle eingefügt werden und nicht nur „irgendwo" im Fenster. Hier muß man bei der Tracking-Funktion die genaue Position suchen und während des Drags auch sinnvoll hervorheben – bei einem Texteditor wird dies wohl üblicherweise der Cursor sein. Autoscrolling ist dann das nächste Problem ...

Spätestens an dieser Stelle empfehle ich das Studium der Apple-Literatur über den Drag & Drop-Manager (das *Macintosh Drag & Drop Developer's Kit*). Diese Dokumentation kann man entweder für viel Geld beim APDA erwerben oder auf *<URL:ftp://ftp.info.apple.com>* finden. Für den bereits erwähnten Toolbox Assistant gibt es ebenfalls bereits eine Datenbank mit allen Drag & Drop-Manager-Funktionen.

Ach ja, eine „Kleinigkeit" noch: keine der Callback-Routinen des Drag & Drop-Managers kann man mit dem CodeWarrior Debugger debuggen! Hier gibt es nur drei Möglichkeiten:

1. fehlerfreien Code erzeugen
2. mit dem MacsBug den 68k-Code auf Assembler-Ebene debuggen
3. sich den teuren (aber sehr guten) Jasiks Debugger besorgen, der ähnlich wie der MacsBug funktioniert, jedoch auch auf Source-Level funktioniert.

Auch wenn das Debugging-Problem auf Anhieb vielleicht nicht so gravierend klingt – es ist das Problem, mit dem man zu kämpfen hat! Ich persönlich nutze übrigens MacsBug und versuche gleich, fehlerfreien Code zu erzeugen ...

Ausblick

Nach diesem – wie immer viel zu kurzen – Einblick in den Drag & Drop-Manager wird im nächsten Monat hoffentlich etwas zum Thema *OpenDoc* erscheinen, welches gerade in der 1.0 Version erschienen ist. Dazu muß ich mir aber selbst erst einmal einen Überblick über OpenDoc verschaffen ...

MFR

```
/* Drag & Drop
 * (c) 1995 MAXON Computer
 * Autor: Markus Fritze
 */

#include <Drag.h>
#include <Gestalt.h>

// unser interner Drag & Drop Typ
const FlavorType  kMyDragItem = ,????';

// Drag und Drop Manager vorhanden?
Boolean   gHasDrag;

/***
 *  Ist der Drag & Drop Manager installiert?
 ***/
void    DragCheck(void)
{
long  reponse;

  if(::Gestalt(gestaltDragMgrAttr, &reponse)
      != noErr)
    return;

  gHasDrag =
    (reponse & (1 << gestaltDragMgrPresent))
      == (1 << gestaltDragMgrPresent);
}

/***
 *  Können wir den Drag akzeptieren?
 ***/
Boolean MyDragIsAcceptable(DragReference
                           theDragRef)
{
  // Anzahl der Items beim Drag
  unsigned short  numItems;
  ::CountDragItems(theDragRef, &numItems);

  // Wir klappern alle Items ab und testen ob
  // wir sie akzeptieren können.
  for(unsigned short index = 1;
        index <= numItems; index++) {
    ItemReference theItemRef;
    OSErr         iErr;

    // Referenz auf ein Item ermitteln
    ::GetDragItemReferenceNumber(theDragRef,
                        index, &theItemRef);

    // Zuerst gucken wir mal nach unserem
    // internen Datentyp
    FlavorFlags   theFlags;
    iErr = ::GetFlavorFlags(theDragRef,
              theItemRef, kMyDragItem,
              &theFlags);
    if(iErr == noErr
        && (theFlags & flavorSenderOnly))
      continue; // => akzeptiert

    // Dateien akzeptieren wir auch
    iErr = ::GetFlavorFlags(theDragRef,
              theItemRef, flavorTypeHFS,
              &theFlags);
    if(iErr == noErr)
      continue; // => akzeptiert

    // Wir haben einen Datentyp gefunden, den wir
    // nicht akzeptieren können => wir
    // akzeptieren den Drag nicht!
    return false;
  }
  return true;
}

/***
 *  Bewegung des Mauszeigers in unseren Fenter(n)
 *  beobachten
 ***/
pascal OSErr  MyDragTracking(
              DragTrackingMessage message,
              WindowPtr theWindow,
              void *handlerRefCon,
              DragReference theDragRef)
{
static Boolean  weAcceptThisItem = false;
static Boolean  inContent = false;

  DragAttributes  flags;
  ::GetDragAttributes(theDragRef, &flags);

  switch(message) {
    case dragTrackingEnterHandler:
          break;

    case dragTrackingEnterWindow:
          weAcceptThisItem =
              MyDragIsAcceptable(theDragRef);
          break;

    case dragTrackingInWindow:
          if(!weAcceptThisItem)
            break;

          Point   mouse;
          GetDragMouse(theDragRef, &mouse, 0L);
          GlobalToLocal(&mouse);
          if(flags & dragHasLeftSenderWindow) {

            // aktiver Bereich unseres Fensters
            Rect  r = theWindow->portRect;
            // Scrollbars abziehen
            r.right -= 15; r.bottom -=15;

            // Mauszeiger im Fenster?
            if(PtInRect(mouse, &r)) {
              RgnHandle hiliteFrame = NewRgn();
              RectRgn(hiliteFrame, &r);
              // blauen Rahmen zeichnen
              ::ShowDragHilite(theDragRef,
                          hiliteFrame, true);
              inContent = true;

            // Mauszeiger gerade aus dem Fenster?
            } else {
              if(inContent) {
                ::HideDragHilite(theDragRef);
                inContent = false;
              }
            }
          }
          // Hier könnten wir eine genaue
          // Position der Maus im Fenster
          // ermitteln, um eine genaue Stelle für
          // das Einfügen des gedroppten Items zu
          // finden und ggf. hervorzuheben.
          break;
```

→

```
139:
140:     case dragTrackingLeaveWindow:
141:          if(weAcceptThisItem &&
142:              (flags & dragHasLeftSenderWindow))
143:            ::HideDragHilite(theDragRef);
144:          break;
145:
146:     case dragTrackingLeaveHandler:
147:          break;
148:   }
149:
150:   return noErr;
151: }
152:
153: /***
154:  *  Drop im Fenster einfügen
155:  ***/
156: pascal OSErr  MyDragReceive(WindowPtr theWindow,
157:                    void *handlerRefCon,
158:                    DragReference theDragRef)
159: {
160:   // Anzahl der Items beim Drag
161:   unsigned short  numItems;
162:   ::CountDragItems(theDragRef, &numItems);
163:
164:   for(unsigned short index = 1;
165:          index <= numItems; index++) {
166:     ItemReference theItemRef;
167:     OSErr         iErr;
168:
169:     // Referenz auf ein Item ermitteln
170:     ::GetDragItemReferenceNumber(theDragRef,
171:                         index, &theItemRef);
172:
173:     // Zuerst gucken wir mal nach unserem
174:     // internen Datentyp
175:     // Wenn wir den erkennen, bedeutet dies, daß
176:     // der Drag innerhalb unseres Programms
177:     // stattfand.
178:     FlavorFlags   theFlags;
179:     iErr = ::GetFlavorFlags(theDragRef,
180:               theItemRef, kMyDragItem,
181:               &theFlags);
182:
183:     if(iErr == noErr &&
184:         (theFlags & flavorSenderOnly)) {
185:
186:       // Wir verschieben unser Item...
187:
188:     } else {
189:       // Der Drag kam von außerhalb!
190:
191:       // Wir akzeptieren in diesem Falle nur HFS
192:       // (= Dateien)
193:       iErr = ::GetFlavorFlags(theDragRef,
194:               theItemRef, flavorTypeHFS,
195:               &theFlags);
196:       if(iErr == noErr) {
197:
198:         // Hiermit können wir abfragen, wie groß
199:         // die zu erwartenden Daten sind
200:         // ok, ok, bei flavorTypeHFS ist das eh
201:         // fix, aber wir könnten ja auch
202:         // einen TEXT-Flavor erwarten.
203:
204:         Size  dataSize;
205:         ::GetFlavorDataSize(theDragRef,
206:               theItemRef, flavorTypeHFS,
207:               &dataSize);
```

```
208:
209:         // Nun holen wir die Daten ab
210:         HFSFlavor   hfs;
211:         ::GetFlavorData(theDragRef, theItemRef,
212:               flavorTypeHFS, &hfs, &dataSize,
213:               0L);
214:
215:         // Hilite Rahmen vom Fenster entfernen,
216:         // da wir wahrscheinlich gleich einen
217:         // teilweisen Redraw machen und dies mit
218:         // Rahmen evtl. Müll ergibt
219:         ::HideDragHilite(theDragRef);
220:
221:         // In „hfs“ steht nun alles, was wir über
222:         // das File wissen müssen.
223:         // Wir können es nun hier öffnen oder was
224:         // auch immer :-)
225:       }
226:     }
227:   }
228:   return noErr;
229: }
230:
231: /***
232:  *  Drag-Callbacks an ein Window anmelden
233:  ***/
234: void    DragInstall(WindowPtr iWind)
235: {
236:   if(!gHasDrag) return;
237:   ::InstallTrackingHandler(MyDragTracking, iWind,
238:                            nil);
239:   ::InstallReceiveHandler(MyDragReceive, iWind,
240:                            nil);
241: }
242:
243: /***
244:  *  vor DisposeWindow() die Drag-Callbacks
245:  *  abmelden
246:  ***/
247: void    DragRemove(WindowPtr iWind)
248: {
249:   if(!gHasDrag) return;
250:   ::RemoveTrackingHandler(MyDragTracking, iWind);
251:   ::RemoveReceiveHandler(MyDragReceive, iWind);
252: }
253:
254: /***
255:  *  Die folgenden Routinen brauchen wir nur als
256:  *  Sender eines Drag:
257:  ***/
258: /***
259:  *  Diese Routine wird angesprungen für Flavors
260:  *  die wir nur auf Anforderung erzeugen. In
261:  *  unserem Beispiel ist es nur der Flavor ‚TEXT‘
262:  ***/
263: pascal OSErr    MyDragSendProc(
264:                     FlavorType theType,
265:                     void *dragSendRefCon,
266:                     ItemReference theItemRef,
267:                     DragReference theDragRef)
268: {
269: MyDocItem   *theItem = (MyDocItem*)theItemRef;
270: OSErr       iErr;
271:
272:   switch(theType) {
273:   case ‚TEXT‘:
274:     Handle  myData =
275:                   MyConvertItemToText(theItem);
276:     ::HLock(myData);
```

→

```
277:     iErr = ::SetDragItemFlavorData(theDragRef,
278:                     theItem, theType, *myData,
279:                     ::GetHandleSite(myData), 0L);
280:     ::DisposeHandle(myData);
281:     break;
282:
283:   default:
284:     iErr = badDragFlavorErr
285:   }
286:   return iErr;
287: }
288:
289: /***
290:  *  Items dem Drag zuordnen
291:  ***/
292: void    MyAddFlavors(DragReference theDragRef,
293:                      WindowPtr theWindow)
294: {
295: MyDocItem   *theItem;
296:
297:   theItem = MyGetFirstSelectedItem(theWindow);
298:   while(theItem) {
299:     // unseren internen Typ übergeben wir direkt
300:     AddDragItemFlavor(theDragRef,
301:             (ItemReference)theItem, kMyDragItem,
302:             theItem->dataPtr, theItem->dataSize,
303:             flavorSenderOnly);
304:
305:     // ‚TEXT' versprechen wir nur. Wir erzeugen
306:     // die Daten nur auf Anforderung
307:     // Dies sollte bei den meisten externen
308:     // Datentypen der Fall sein.
309:     AddDragItemFlavor(theDragRef,
310:         (ItemReference)theItem, 'TEXT', 0L, 0L,
311:         0);
312:
313:     theItem = theItem->next;
314:   }
315: }
316:
317: /***
318:  *  Eine Region mit den Rahmen aller unserer
319:  *  Items
320:  ***/
321: void    MyGetDragRegion(DragReference theDragRef,
322:                         WindowPtr theWindow,
323:                         RgnHandle dragRegion)
324: {
325: MyDocItem   *theItem;
326: RgnHandle   tempRgn = NewRgn();
327: Point       globalPt = {0,0};
328:
329:   // globale Koordinate vom Window-Ursprung
330:   // errechnen
331:   LocalToGlobal(&globalPt);
332:
333:   theItem = MyGetFirstSelectedItem(theWindow);
334:   while(theItem) {
335:     // Region von einem Item
336:     CopyRgn(theItem->theRegion, tempRgn);
337:     // die Region um einen Pixel vergrÖßern
338:     InsetRgn(tempRgn, 1, 1);
339:     // und von der originalen Region abziehen
340:     DiffRgn(theItem->theRegion, tempRgn,
341:                     tempRgn);
342:     // schon haben wir einen Rahmen!
343:
344:     // in globale Koordinaten umrechnen
345:     OffsetRgn(tempRgn, globalPt.h, globalPt.v);
346:
347:     // zu den restlichen Items mergen
348:     UnionRgn(tempRgn, dragRegion, dragRegion);
349:
350:     // Außenmaße von de Region an den Drag
351:     // Manager melden
352:     SetDragItemBounds(theDragRef,
353:                       (ItemReference)theItem,
354:                       &(*tempRgn)->rgnBBox);
355:
356:     theItem = theItem->next;
357:   }
358:   DisposeRgn(tempRgn);
359: }
360:
361: /***
362:  *  Ein Drag wurde gestartet
363:  ***/
364: void    MyStartDrag(EventRecord *theEvent,
365:                     WindowPtr theWindow)
366: {
367: DragReference   theDragRef;
368:
369:   ::NewDrag(&theDragRef);
370:   MyAddFlavors(theDragRef, theWindow);
371:   RgnHandle dragRegion = ::NewRgn();
372:   MyGetDragRegion(theDragRef, theWindow,
373:                   dragRegion);
374:   ::SetDragSendProc(theDragRef, MyDragSendProc,
375:                     0L);
376:   ::TrackDrag(theDragRef, theEvent, dragRegion);
377:   ::DisposeRgn(dragRegion);
378:   ::DisposeDrag(theDragRef);
379: }
380:
381: /***
382:  *
383:  ***/
384: void    MyMouseDown(EventRecord *theEvent)
385: {
386: WindowPtr theWindow;
387: short     thePart = ::FindWindow(theEvent->where,
     &theWindow);
388:
389:   switch(thePart) {
390:   case inContent:
391:       if(theWindow != ::FrontWindow()) {
392:         ::SelectWindow(theWindow);
393:       } else {
394:         // Mausklick normal auswerten. Neu:
395:         // Testen ob der Klick auf
396:         // ein dragbares Objekt erfolgte
397:         Boolean isOnItem =
398:             MyContentClick(theEvent, theWindow);
399:
400:         // Drag starten, wenn Maus bewegt wurde:
401:         if(isOnItem && gHasDrag &&
402:             ::WaitMouseMoved(theEvent->where)) {
403:           MyStartDrag(theEvent, theWindow);
404:         }
405:       }
406:       break;
407:
408:   // weitere Standard-Windowverwaltung...
409:   }
410: }
```