Newton-Keyboard am Macintosh

Faceless Background Applications

Diesen Monat möchte ich etwas zu dem Anschluß eines Newton-Keyboards am Mac schreiben. Das Newton-Keyboard ist nämlich durch seine kleinen Abmessungen ideal für Messen etc. zu verwenden.

Das Newton-Keyboard ist für etwas über 100,– DM im Apple Fachhandel zu erhalten. Das kleine schwarze Keyboard wird inklusiver einer Tasche geliefert. Da es beim Newton an die serielle Schnittstelle angeschlossen wird, kam mir die Idee, das Keyboard an einen Mac anzuschließen. Mit Hilfe eines Terminalprogramms habe ich schnell ermitteln können, daß das Keyboard mit 9600 Baud seine Daten an den Rechner sendet – 8 Bits pro Zeichen, keine Parität, ein Stopp-Bit. Elektrisch war die Ansteuerung also kein Problem.

Nun mußte ich rausbekommen, was für Daten von der Tastatur gesendet werden. Auch hier: keine Überraschungen. Die Tastatur sendet bei jedem Tastendruck bzw. beim Loslassen einer Taste ein Byte. Unterscheiden kann man das Drücken bzw. Loslassen einer Taste daran, daß beim Loslassen Bit 7 gesetzt wird. Der 7-Bit-Scancode entspricht quasi zu 100% denen der normalen Mac-Keyboards.

Sobald die Schnittstelle geöffnet wird, sendet das Keyboard einen String mit Identifikationsdaten. Leider habe ich keine genaueren Informationen über den Aufbau der Daten, es reicht jedoch zur Erkennung der Newton-Tastatur.

Die Software

Die Aufgabe eines Newton-Keyboard-Treibers ist also recht klar umrissen: Serielle Schnittstelle öffnen, die ersten Bytes ignorieren, dann bei jedem gesendeten Byte entweder das Drükken oder das Loslassen einer Taste melden. So weit, so gut. Nur wie macht man das?

Ich wollte versuchen, keine Systemerweiterung zu schreiben, die irgendwelche Traps patcht, sondern dachte zuerst: das muß ja einfach sein, einfach per *PostEvent()* das Drücken und Loslassen melden und fertig. Leider ist es „etwas" komplizierter ...

Der einfache Teil ist klar: wir schreiben eine „Faceless Background Application" kurz FBA. Eine FBA verhält sich für den normalen Anwender wie eine Systemerweiterung und wird auch in dem Ordner installiert. Für das MacOS handelt es sich aber um ein ganz normales Mac-Programm mit Event-Loop. FBAs dürfen jedoch – mit Ausnahme des Notification Managers – keine Bildschirmausgaben tätigen. Für einen Tastaturtreiber stellt dies aber kein Problem dar. FBAs erlauben AppleEvents, und so kann man sehr elegant kleinere Erweiterungen für viele Programme als FBA implementieren z.B. AppleScript-Erweiterungen für FileMaker Pro.

Die serielle Schnittstelle

Das Öffnen der Schnittstelle mit *OpenDriver()* ist relativ einfach und klar. Die Namen der Ports habe ich in der Resource-Fork abgelegt, so daß man sie bei Bedarf leicht ändern kann. Besser wäre natürlich ein Kontrollfeld, mit dem man den Port direkt wählen kann, oder die Möglichkeit den Port automatisch zu ermitteln. Darauf haben wir hier einmal verzichtet. Wer möchte, kann sich so etwas ja selbst programmieren. Die Baud-Rate setzen wir danach auf 9600 8N1 – ist normalerweise sowieso Standard, aber sicher ist sicher. Handshaking ist für die Tastatur nicht nötig. Um der Tastatur die Möglichkeit zu geben, ihre Identifikation zu senden, wartet der Treiber eine halbe Sekunde und liest dann alle an der seriellen Schnittstelle anliegenden Bytes ein. Welche Bytes erwartet werden, kann man dem Listing entnehmen. Ein kleiner Plausibilitätstest stellt sicher, daß auch wirklich eine Newton-Tastatur angeschlossen ist.

Faceless Background Applications

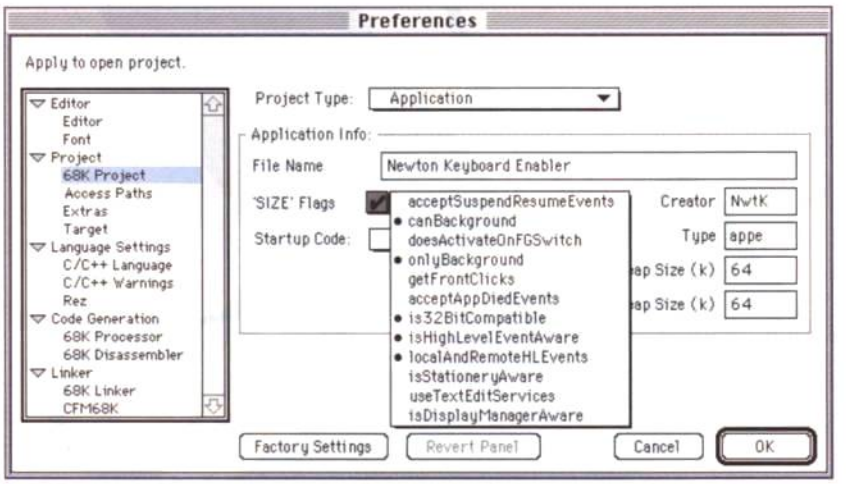
Nun kommt aber der Trick: Anstatt im Event-Loop die serielle Schnittstelle zu pollen, lesen wir asynchron von der Schnittstelle! Dies belastet das MacOS fast gar nicht, und bei jedem Byte, das an der Schnittstelle ankommt, wird automatisch eine Callback-Routine aufgerufen. Dort verarbeiten wir das gelesene Byte und starten den nächsten Lesevorgang. Einzige Falle ist hier, daß man den Parameterblock und andere Variablen global halten muß, denn auf dem lokalen Stack würden sie ja beim Verlassen des Unterprogramms gleich gelöscht werden. Abstürze wären die Folge.

In der Testphase hat es sich als nützlich erwiesen, wenn man den Treiber beenden kann. Dazu setzt man den Dateityp nicht auf „appe" (der Filetype von FBAs), sondern ganz normal auf „APPL". Nun kann man den Treiber einfach per Doppelklick starten. Per Hot-Key (Control-Option-^) bzw. Quit-AppleEvent verläßt der Treiber nun den Event-Loop. Nach drei *SysBeeps()* wartet er auf ein letztes Byte an der seriellen Schnittstelle und beendet sich dann selbst. Den Hot-Key muß man etwas länger drücken, denn der Event-Loop wird nur einmal pro Sekunde durchlaufen. Normalerweise braucht der Treiber ja diesen Event-Loop gar nicht, und so kostet er wenigstens kaum Rechenzeit.

Die Scancodes

Die Tastatur meldet nur ein Byte für einen Tastendruck. Dieses Byte (Scancode genannt) muß in einen ASCII-Code umgerechnet werden. Dazu gibt es im MacOS die KMAP- und KCHR-Resourcen. KMAP erlaubt es Scancodes von verschiedenen Tastaturen zu normalisieren, d.h. auf gleiche Scancodes umzurechnen. Jede Tastatur kann schließlich völlig eigene Scancodes senden. Ferner wird in der KMAP-Resource definiert, bei welchen Tastendrücken die LEDs am erweiterten Keyboard eingeschaltet werden. Das Newton-Keyboard hat keine LEDs, so erübrigt sich weiterer Aufwand in dieser Richtung.

Ebenfalls muß das Verhalten der Capslock-Taste simuliert werden. Beim Newton-Keyboard wird sie ja nicht mechanisch blockiert. Man muß also per Software einen Caps-Schalter implementieren.



Hier die eingestellten „SIZE"-Flags für unsere Faceless Background Application

Den erzeugten normalisierten Keycode muß man nun in die KeyMap-Bit-Maske übertragen. Diese Bit-Maske erhält man, wenn man die Funktion *GetKeys()* aufruft. Viele Programme tun dies, und wir müssen diese Bit-Maske selbst aktualisieren. Hierbei gibt es folgendes zu beachten: es können mehrere Tastaturen angeschlossen sein! Jede Tastatur braucht ihre eigene interne KeyMap, die bei jedem Tastendruck einfach in die KeyMap des Systems kopiert wird.

Ferner müssen noch einige (undokumentierte) globale Variablen richtig gesetzt werden, damit u.a. Auto-Repeat funktioniert. Diese Variablen habe ich durch Analyse des Original-ADB-Treibers von Apple gefunden. Nun ja, eine andere Möglichkeit kenne ich nicht, und Auto-Repeat ist irgendwie zu wichtig, als das man es nicht unterstützt.

Zu guter Letzt wird der Scancode in einen (oder zwei) ASCII-Code(s) umgerechnet. Dazu braucht man lediglich die Funktion *KeyTranslate()* aufrufen. Diese Funktion bekommt eine KCHR-Routine übergeben, welche beschreibt, wie man einen Scancode in einen ASCII-Code umrechnet. Unterschiedliche KCHR-Resourcen erlauben es, z.B. für verschiedene Sprachen verschiedene Tastaturlayouts zu unterstützen. Die Scancodes sind nämlich – trotz unterschiedlich beschrifteter Tastenkappen – gleich! Es gibt aber noch ein paar Dinge zu beachten: Die Funktion kann zwei ASCII-Codes erzeugen, da einige Tastaturen auf einen realen Tastendruck zwei Tastendrükke simulieren. Ich glaube in Japan ist so etwas nicht unüblich. Ferner muß man den Status „state" global halten. In diesem Status wird zwischen zwei Tastendrücken gespeichert, ob z.B. ein Dead-Key gedrückt wurde. Beispiel: man drückt ein Taste, die ein Akzent einleitet, und danach eine Taste, auf welche das Akzent gesetzt werden soll.

Zu guter Letzt wird der Tastendruck als *keyDown*- bzw. *keyUp*-Message an das MacOS gepostet. Natürlich nicht ohne vorher noch ein paar undokumentierte Variablen im MacOS zu setzen. Nun ja ...

Die Compilierung

Um das Programm zu compilieren, muß lediglich der Source wie auch die Resource compiliert werden. Das Programm bekommt als Dateityp „appe" und als Creatorcode „NwtK" zugewiesen. 64k Speicherzuteilung dürften ausreichend sein. Lediglich die Flags müssen sorgfältig gesetzt werden (siehe Grafik). Wer will, kann den Dateityp auf „APPL" setzen und den Treiber per Doppelklick bei Bedarf starten.

Um die serielle Schnittstelle zu ändern, muß man lediglich in der „STR# 128"-Resource die Einträge von „.AIn" und „.AOut" (Modemanschluß) z.B. auf „.BIn" und „.BOut" (Druckeranschluß) ändern.

In der „PREF 128"-Resource kann man auch das Keyboard-Layout umschalten: das letzte Byte hat den Wert

3 für Deutschland oder 0 für US-Englisch. Weitere Layouts sind möglich, man muß lediglich entsprechende KCHR-Resourcen zur Verfügung stellen. Üblichweise sind seit Mac OS 7.5 alle internationalen Keyboard-Layouts bereits im System vorhanden.

Das erste Byte ermöglicht es, zwischen der KMAP-0- und KMAP-1-Resource zu wählen. Diese Layouts unterscheiden sich durch die Behandlung der gemeinsamen Shift-Tasten. Üblich ist es, daß beide Shift-, Command-, etc.-Tasten den gleichen Scancode an die Anwenderprogramme melden. Wählt man die KMAP-1-Resource, werden diese Tasten unterschiedlich gemeldet. Programme können dies zwar per Software umschalten (MagicMac nutzt dies z.B.), aber dies erfordert die direkte Programmierung der ADB-Tastaturen, was bei unserer Newton-Tastatur an der seriellen Schnittstelle halt nicht möglich ist.

Im abgedruckten Listing fehlt aus Platzgründen die Resource. Natürlich können Sie – wie immer – das komplette Projekt inkl. Sourcen und Compilate von dem FirstClass-Server der MacOPEN (06196-484457) downloaden. Die jeweils aktuelle Version des Newton-Keyboard-Enablers findet sich im Internet auf *http://www.emagic.de/mmm/* – neben vielen weiteren Programmen vom Autor. Viel Spaß also mit der Newton-Tastatur am Mac!

MFR

```
/***
 *  Newton Keyboard Enabler.c
 *
 *  Erlaubt die Nutzung eines seriellen Newton-
 *  Keyboards an einem Mac. Das Keyboard verhält
 *  sich in fast allen Fällen genau wie ein
 *  originales ADB-Keyboard (Ausnahme: MacsBug
 *  kann es nicht nutzen)
 *
 * Entwickelt mit dem CodeWarrior 9 von
 * Metrowerks.
 *
 *  (c)1996 MAXON Computer, Markus Fritze
 ***/

// true, wenn das Programm beendet werden soll
Boolean gDoQuitFlag;

/***
 *  unsere AppleEvent-Routinen
 *  (schließlich sind wir ein ordentliches
 *  MacOS Programm)
 ***/
static pascal OSErr DoAENoErr(
    const AppleEvent*, AppleEvent*, long)
{
  return noErr;   // AppleEvent ist ok
}

static pascal OSErr DoAEQuitAppl(
    const AppleEvent*, AppleEvent*, long)
{
  gDoQuitFlag = true; // Programm beenden
  return noErr;
}

// einen (hoffentlich) undefinierten Code
// benutzen wir als ID-Code für die Tastatur
#define NEWTON_KEYBOARD_CODE    117L

// Zugriffsfunktionen ähnlich <LowMem.h>
// für den Tastaturtreiber
static inline SInt16 LMGetKeyLast()
        { return *(SInt16*)0x0184; };
static inline void LMSetKeyLast(SInt16 value)
        { *(SInt16*)0x0184 = value; };
static inline SInt16 LMGetHiKeyLast()
        { return *(SInt16*)0x0216; };
static inline void LMSetHiKeyLast(SInt16 value)
        { *(SInt16*)0x0216 = value; };

static inline SInt32 LMGetKeyTime()
        { return *(SInt32*)0x0186; };
static inline void LMSetKeyTime(SInt32 value)
        { *(SInt32*)0x0186 = value; };
static inline SInt32 LMGetKeyRepTime()
        { return *(SInt32*)0x018A; };
static inline void LMSetKeyRepTime(SInt32 value)
        { *(SInt32*)0x018A = value; };

// ohne "inline", wegen eines 68k Compilerbugs
// beim CodeWarrior 9
static /*inline*/ KeyMap *LMGetKeyMapPtr()
        { return (KeyMap*)0x0174; };

// Unsere globalen Variablen für die Tastatur
Handle      gKMAP;
Handle      gKCHR;
UInt8     gKeyMap[16];

/***
 *  Keyboard-Variablen initialisieren
 ***/
static void InitKeyboard()
{
  Handle  thePref =
    ::Get1Resource('PREF', 128);

  // eigener Typ: Newton Keyboard
  gKMAP = ::Get1Resource('KMAP', **thePref);
  if(!gKMAP) ::ExitToShell();
  ::HLockHi(gKMAP);

  // ein deutsches Keyboard:
  gKCHR = ::GetResource('KCHR',
          ((short*)*thePref)[1]);
  if(!gKCHR)
    // ein US-Keyboard:
    gKCHR = ::GetResource('KCHR', 0);
  if(!gKCHR) ::ExitToShell();
  ::HLockHi(gKCHR);

  // eigene Keymap löschen
  for(int i=0; i<sizeof(gKeyMap); i++)
    gKeyMap[i] = 0;

  ::ReleaseResource(thePref);
}

/***
 *  Tastencode senden
 ***/
static void PostKeyMessage(
        UInt8 inKey, UInt8 inKeyCode)
{
  // keine Taste => raus
  if(inKey == 0x00L) return;

  // Message zusammensetzen
  UInt32  theMessage = inKey
        | UInt16(inKeyCode << 8)
        | (NEWTON_KEYBOARD_CODE << 16);

  // Taste gedrückt
  if(!(inKeyCode & 0x80)) {
```

```
116:     SInt32  theTicks = LMGetTicks();
117:     LMSetKeyTime(theTicks);
118:     LMSetKeyRepTime(theTicks);
119:     LMSetKeyLast(theMessage);
120:     LMSetHiKeyLast(NEWTON_KEYBOARD_CODE);
121:     ::PostEvent(keyDown, theMessage);
122:
123:   // Taste losgelassen
124:   } else {
125:     // Key-Up-Flag löschen
126:     theMessage &= 0xFFFF7FFF;
127:     ::PostEvent(keyUp, theMessage);
128:   }
129: }
130:
131: /***
132:  *  Tastendruck (bzw. das Loslassen) dem MacOS
133:  *  melden
134:  ***/
135: static void EnterKeycode(UInt8 inCode)
136: {
137:   // aktuelle Taste im System löschen
138:   LMSetKeyLast(0);
139:   LMSetHiKeyLast(0);
140:
141:   // true, wenn Taste losgelassen wurde
142:   Boolean theDownFlag =
143:                 (inCode & 0x80) == 0x80;
144:
145:   // MacOS-Keycode erzeugen
146:   UInt8 theKeyCode;
147:   Ptr   theKMAP = *gKMAP;
148:   theKeyCode = theKMAP[(inCode & 0x7F) + 4];
149:   // Sondercode erkannt?
150:   if(theKeyCode & 0x80) {
151:
152:     // erstmal das Kennungs-Bit löschen
153:     theKeyCode &= 0x7F;
154:
155:     // Anzahl der Sondereinträge
156:     SInt16  theCount =
157:       *reinterpret_cast<SInt16*>
158:             (&theKMAP[0x84]);
159:
160:     // ab hier geht es mit den Tabellen los
161:     UInt8 *theKMapP =
162:       reinterpret_cast<UInt8*>
163:             (&theKMAP[0x86]);
164:     while(theCount-- > 0) {
165:       // Code gefunden?
166:       if(*theKMapP++ != theKeyCode) {
167:         // zum nächsten Eintrag
168:         theKMapP += theKMapP[1] + 2;
169:         continue;
170:       }
171:       if((*theKMapP & 0x0F) == 0x00)
172:         return;
173:       break;
174:     }
175:   }
176:
177:   // Capslock Abfrage
178:   if(theKeyCode == 0x39) {
179:     if(theDownFlag) { // Taste gedrückt?
180:
181:       // Caps bereits gesetzt?
182:       if(gKeyMap[theKeyCode >> 3]
183:         & (1 << (theKeyCode & 7))) {
184:         // dann lösen!
185:         theDownFlag = false;
186:       }
187:     } else {  // Taste losgelassen?
188:       // (das interessiert uns nie!)
189:       return;
190:     }
191:   }
192:
193:   // in die KeyMap eintragen (vorerst nur in
194:   // die eigene)
195:   if(theDownFlag) {
196:     gKeyMap[theKeyCode >> 3] |=
197:         1 << (theKeyCode & 7);
198:   } else {
199:     gKeyMap[theKeyCode >> 3] &=
200:         ~(1 << (theKeyCode & 7));
201:
202:     // Flag für "losgelassen"
203:     theKeyCode |= 0x80;
204:   }
205:
206:   // Tastencodes in globalen Variablen merken
207:   LMSetKbdLast(theKeyCode);
208:   LMSetKbdType(NEWTON_KEYBOARD_CODE);
209:
210:   // globale KeyMap updaten
211:   ::BlockMoveData(gKeyMap, LMGetKeyMapPtr(),
212:           sizeof(KeyMap));
213:
214:   // aktuelle Modifiers für KeyTranslate lesen
215:   UInt16  theModifiers = *(3 +
216:           reinterpret_cast<UInt16*>
217:           (LMGetKeyMapPtr()));
218:
219:   // ROL.W #1,<ea>
220:   theModifiers = (theModifiers >> 15)
221:         | (theModifiers << 1);
222:
223:   // ASCII-Codes (denkbar: zwei pro
224:   // Tastendruck!) errechnen
225:   static UInt32 state = 0;
226:   UInt32  lStructure = ::KeyTranslate(*gKCHR,
227:         theKeyCode | (theModifiers << 8),
228:                     &state);
229:
230:   // ggf. zwei Tasten posten
231:   PostKeyMessage(lStructure >> 16, theKeyCode);
232:   PostKeyMessage(lStructure, theKeyCode);
233: }
234:
235: /***
236:  *  diese asynchrone Routine pollt das Keyboard
237:  *  an der Seriellen
238:  ***/
239: #include <Serial.h>
240:
241: // UPP für die Callback-Routine
242: IOCompletionUPP gIOUPP;
243:
244: // Refnums für Serial ein/aus
245: SInt16      gSDIn, gSDOut;
246:
247: // das empfangene Zeichen
248: UInt8     gInChar;
249:
250: // der Parameterblock (asynchron!)
251: ParamBlockRec gParamBlk;
252:
253: /***
254:  * das nächste Byte von der
255:  * Tastatur asynchron lesen
256:  ***/
257: static void   GetNextByte()
258: {
259:   if(gDoQuitFlag) return;
260:   // Callback setzen
261:   gParamBlk.ioParam.ioCompletion = gIOUPP;
262:   // Port lesen
263:   gParamBlk.ioParam.ioRefNum = gSDIn;
```

```
264:    // Buffer auf unser Byte
265:    gParamBlk.ioParam.ioBuffer = (Ptr)&gInChar;
266:     // ein Byte lesen
267:    gParamBlk.ioParam.ioReqCount = 1L;
268:    // ab der aktuellen Position
269:    gParamBlk.ioParam.ioPosMode = fsAtMark;
270:    // kein Offset...
271:    gParamBlk.ioParam.ioPosOffset = 0L;
272:    // Anforderung absetzen
273:    PBReadAsync(&gParamBlk);
274: }
275:
276: /***
277:  * Diese Routine wird angesprungen,
278:  * wenn ein Byte eingetroffen ist.
279:  ***/
280: static void   MyCompletion(
281:           ParmBlkPtr ioParam : __A0)
282: {
283: #pragma unused(ioParam)
284:
285:    // Byte verarbeiten
286:    EnterKeycode(gInChar);
287:
288:    // nächstes Byte anfordern
289:    GetNextByte();
290: }
291:
292: /***
293:  *  main()
294:  ***/
295: void    main()
296: {
297:    // 16k anstatt 2k an Stack!
298:    ::SetApplLimit((Ptr)((UInt32)
299:           ::GetApplLimit() - 0x4000));
300:
301:    // Crasht vor MacOS 7.5.4, falls eine zweite
302:    // FBA ebenfalls MaxApplZone() aufruft:
303:    // ::MaxApplZone();
304:
305:    // weitere Init-Calls sind bei FBAs nicht
306:    // erlaubt
307:    ::InitGraf(&qd.thePort);
308:
309:    // AppleEvents installieren (wenn vorhanden)
310:    long  response;
311:    if(!::Gestalt(gestaltAppleEventsAttr,
312:           &response)) {
313:      if(response &
314:        (1L<<gestaltAppleEventsPresent)) {
315:
316:        if(::AEInstallEventHandler(
317:              kCoreEventClass,
318:              kAEOpenApplication,
319:              NewAEEventHandlerProc(DoAENoErr),
320:              0L, 0))
321:          return;
322:
323:        if(::AEInstallEventHandler(
324:              kCoreEventClass,
325:              kAEOpenDocuments,
326:              NewAEEventHandlerProc(DoAENoErr),
327:              0L, 0))
328:          return;
329:
330:        if(::AEInstallEventHandler(
331:              kCoreEventClass,
332:              kAEPrintDocuments,
333:              NewAEEventHandlerProc(DoAENoErr),
334:              0L, 0))
335:          return;
336:
337:        if(::AEInstallEventHandler(
338:              kCoreEventClass,
339:              kAEQuitApplication,
340:              NewAEEventHandlerProc(DoAEQuitAppl),
341:              0L, 0))
342:          return;
343:      }
344:    }
345:
346:    // globale Keyboard-Variablen initialisieren
347:    InitKeyboard();
348:
349:    // ".AIn" und ".AOut" öffnen
350:    OSErr theErr;
351:    Str255  theStr;
352:    ::GetIndString(theStr, 128, 2);
353:    theErr = ::OpenDriver(theStr, &gSDOut);
354:    if(theErr) ::ExitToShell();
355:    ::GetIndString(theStr, 128, 1);
356:    theErr = ::OpenDriver(theStr, &gSDIn);
357:    if(theErr) goto raus;
358:
359:    // 9600 8N1
360:    theErr = ::SerReset(gSDOut,
361:                baud9600+data8+stop10+noParity);
362:    if(theErr) goto raus;
363:
364:    // Handshaking ausschalten
365:    SerShk  theSHandShk;
366:    theSHandShk.fXOn = 0;
367:    theSHandShk.fCTS = 0;
368:    theSHandShk.errs = 0;
369:    theSHandShk.evts = 0;
370:    theSHandShk.fInX = 0;
371:    theSHandShk.fDTR = 0;
372:    theErr = ::Control(gSDOut, 14, &theSHandShk);
373:    if(theErr) goto raus;
374:
375:    long  theTicks;
376:    // 1/2 Sekunde auf das Keyboard warten
377:    ::Delay(30, &theTicks);
378:
379:    // Anzahl der Byte an der Schnittstelle
       ermitteln
380:    SInt32  theCount;
381:    ::SerGetBuf(gSDIn, &theCount);
382:
383:    // und alle lesen
384:    Str255  theBuf;
385:    ::FSRead(gSDIn, &theCount, &theBuf);
386:
387:    // Daten von der Tastatur zum Rechner, wenn die
388:    // Schnittstelle angeschaltet wird (9600 8N1):
389:    //  <0x16><0x10> 0x02,
390:    //  'd_id', 0x0CL,          // Device-ID?
391:    //  'kybd','appl', 0x01L,   // Keyboard-Typ
392:    //  'nofm', 0L, 0x1003dde7L   // ???
393:    if(reinterpret_cast<long*>(&theBuf)[3]
394:          != 'ybda')
395:      goto raus;
396:
397:    gIOUPP = NewIOCompletionProc(MyCompletion);
398:    GetNextByte();  // erstes Byte erwarten
399:
400:    gDoQuitFlag = false;
401:    while(!gDoQuitFlag) {
402:      EventRecord theEvent;
403:      // nur einmal pro Sekunde erwarten wir einen
404:      // Null-Event!
405:      ::WaitNextEvent(
406:             everyEvent, &theEvent, 60, 0L);
407:      if(theEvent.what == kHighLevelEvent)
408:        ::AEProcessAppleEvent(&theEvent);
409:
410: #if DEBUG
```

```
411:     // zum Debuggen: '^' + Control + Option
412:     // beendet das Programm!
413:     KeyMap  theMap;
414:     ::GetKeys(theMap);
415:     if((theMap[0] & 0x40000) &&
416:         ((theMap[1] & 0xC) == 0xC)) {
417:       break;
418:     }
419: #endif
420:   }
421:   // auf ein letztes Byte warten!
422:   SysBeep(10); SysBeep(10); SysBeep(10);
423:
424:   // auf Abschluß des aktuellen Polls warten
425:   while(gParamBlk.ioParam.ioResult > 0) {}
426:
427:   // Tastaturstatus zurücksetzen
428:   LMSetKeyLast(0);
429:   LMSetHiKeyLast(0);
430:   for(int i=0; i<sizeof(gKeyMap); i++)
431:     gKeyMap[i] = 0;
432:   ::BlockMoveData(gKeyMap, LMGetKeyMapPtr(),
433:                   sizeof(KeyMap));
434:
435: raus:
436:   if(gSDOut) ::KillIO(gSDOut);
437:   if(gSDIn) ::CloseDriver(gSDIn);
438:   if(gSDOut) ::CloseDriver(gSDOut);
439: }
440:
441: /***
442:  * Newton Keyboard.r
443:  ***/
444: resource 'KMAP' (0) {
445:    0,
446:    0,
447:    {  0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,
448:       18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,
449:       33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,
450:       48,49,50,51,52,53,59,55,56,57,58,59,56,58,59,
451:       63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,
452:       78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,
453:       93,94,95,96,97,98,99,100,101,102,103,104,105,
454:       106,107,108,109,110,111,112,113,114,115,116,
455:       117,118,119,120,121,122,123,124,125,126,127
456:    },
457:    {
458:    }
459: };
460:
461: resource 'KMAP' (1) {
462:    0,
463:    0,
464:    {  0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,
465:       18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,
466:       33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,
467:       48,49,50,51,52,53,59,55,56,57,58,59,60,61,62,
468:       63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,
469:       78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,
470:       93,94,95,96,97,98,99,100,101,102,103,104,105,
471:       106,107,108,109,110,111,112,113,114,115,116,
472:       117,118,119,120,121,122,123,124,125,126,127
473:    },
474:    {
475:    }
476: };
477:
478: resource 'STR#' (128, "Portnames") {
479:    {  ".AIn",
480:       ".AOut"
481:    }
482: };
483:
484: data 'TMPL' (128, "PREF") {
485:    /* .Different Shift */
486:    $"1544 6966 6665 7265 6E74 2053 6869 6674"
487:    /* -Keys?BOOL.Keybo */
488:    $"2D4B 6579 733F 424F 4F4C 0E4B 6579 626F"
489:    /* ardregionRGNC */
490:    $"6172 6472 6567 696F 6E52 474E 43"
491: };
492:
493: data 'PREF' (128) {
494:    $"0000 0003"
495: };
```