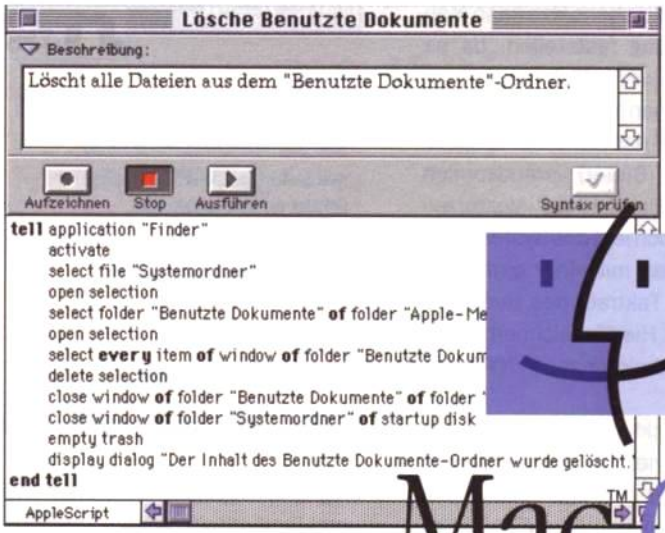


Apple Events

Scripting mit dem MacOS

Dem MacOS wurde immer vorgeworfen, daß ihm die Möglichkeiten einer internen Script-Sprache fehlen. Unter UNIX oder DOS hingegen kann man leicht sich wiederholende Aktionen in Scripts zusammenfassen. Dies ist unter einem grafischen Betriebssystem auch nicht gerade einfach, denn das einfache Aufzeichnen von Mausklicks und Tastendrücken kann kaum befriedigen.

Ich will hier jedoch nicht auf die Programmierung in der Sprache AppleScript eingehen, sondern vielmehr auf die Implementation von der AppleScript-Fähigkeit in eigenen Programmen. Zudem ist ein gewisses Verständnis von Apple Events sehr wichtig, schließlich benutzt nicht nur AppleScript Apple Events, sondern z.B. auch der Edition Manager und Apple Guide. Apple Events und AppleScript setzen System 7 oder neuer voraus. Dies muß man mit *Gestalt()* abfragen, bevor man diese Funktionen in Programmen nutzt!

High Level Events

In der untersten Ebene hat Apple mit System 7 die sogenannten High Level Events eingeführt. Hierbei handelt es sich um bis zu 64kB große Datenblöcke, die von einem Programm an ein anderes verschickt werden können. Man kann diese High Level Events auch an Programme auf fremden Rechnern über das Netzwerk schicken.

Die High Level Events bekommt man als Programm ganz normal mit *WaitNextEvent()* gemeldet. Der Empfänger muß – wie bei einer Commandline – allerdings genau wissen, was in dem Datenblock enthalten ist, den er gerade erhalten hat. Jeder High Level Event kann jedoch über seine Event-ID und die Event-Klasse identifiziert werden – ganz ähnlich dem Prinzip von Filetype und Creator bei Dateien. Einige Event-Klassen für Apple Events sind bereits vordefiniert (die Core Event Class, die Core Suite sowie die Finder Events und die Events vom Edition Manager). Im Gegensatz zu einer Commandline kann man einen High Level Event natürlich nicht nur beim Programmstart, sondern jederzeit erhalten.

Apple Events

Oberhalb der High Level Events – als eine spezielle Klasse von High Level Events – hat Apple die sogenannten Apple Events definiert. Apple Events unterliegen einer genau definierten Struktur, so daß Probleme, wie sie beim Parsen einer Commandline auftreten können, ausgeschlossen sind. Parameter werden vom Empfänger anhand einer 4-Byte-Kennung identifiziert – die Reihenfolge spielt somit keine Rolle. Ferner können Parameter als optional deklariert werden. Auch eine automatische Antwort an den Absender des Events ist möglich. Diese Antwort enthält normalerweise einen Fehlercode, jedoch kann man auch noch beliebige andere Daten übergeben.

Apple hat sehr viele Datentypen für die Parameter bereits definiert, so daß man sich normalerweise keine eigenen Typen ausdenken muß. So kann man neben den üblichen Integern, Strings, Boolean-Werten etc. auch Aliases und Listen von Typen übergeben. Selbst komplexe geschachtelte Typen sind möglich! Viele eventuelle Typenwandlungen, z.B. einer Integerzahl in einen String, übernimmt der Apple Event Manager automatisch. Weitere Wandlungen kann man jederzeit selbst definieren.

Üblicherweise verschicken übrigens alle Programme nur Events im Apple-Event-Format, also keine eigenen High Level Events. Es macht einfach keinen Sinn, den sehr mächtigen Standard nicht zu nutzen. Zudem bauen die weiteren Schichten natürlich auf Apple Events auf.
Die bekanntesten Apple Events sind die 4 required Events der Core Event Class, die jedes System-7-kompatible Programm unterstützen muß:

kAEOpenApplication
Ein Programm wurde gestartet. Viele Programme öffnen bei diesem Event ein leeres Dokument.

kAEOpenDocuments
Ein Programm wurde gestartet, indem Dokumente dieses Programms geöffnet wurden. Als Parameter wird eine Liste der Dokumente übergeben.

kAEPrintDocuments
Wie bei *kAEOpenDocuments* wird das Programm gestartet und eine Liste an Dokumenten übergeben. Diese werden jedoch nicht geöffnet, sondern gedruckt. Der Finder schickt nach einem Print Event üblicherweise auch gleich einen Quit Event.

kAEQuitApplication
Das Programm soll beendet werden. Wird z.B. beim Neustart verschickt, damit vorher alle Programme ordentlich beendet werden. Meldet ein Programm einen Fehler (z.B. –128, d.h. User hat abgebrochen), wird der Neustart verhindert.

Wer den Edition Manager (Herausgeben und Abonnieren) unterstützen will, muß zudem noch ein paar weitere Apple Events unterstützen.

OSA, AppleScript

OSA ist die Abkürzung für die **O**pen **S**cripting **A**rchitecture, die eine Schnittstelle zwischen Script-Sprachen und den Anwenderprogrammen darstellt. So gibt es neben AppleScript z.B. auch noch die Sprache „The Frontier". Das scriptfähige Anwenderprogramm braucht allerdings gar nichts von der Script-Sprache zu kennen. Gleiches gilt auch für die Script-Sprache: diese kennt natürlich ebenfalls nicht jedes Anwenderprogramm, welches Scripting unterstützt.

AppleScripts kann man mit dem AppleScript Editor von Apple aufzeichnen und editieren. Generell sollte man sich zumindest rudimentäre Kenntnisse von der Sprache AppleScript aneignen, da es das Programmieren von scriptable-Programmen doch erheblich erleichtert.

scriptable
Durch das einfache Hinzufügen einer 'aete'-Resource (**A**pple **E**vents **T**erminology **E**xtension) kann man sein Programm nun schon scriptable machen. Diese Resource ordnet die Befehlswörter für AppleScript den Apple Events zu. Dazu gehören auch die Beschreibungen der Parameter. Dies reicht häufig schon für einfache Implementierungen.

'aete'-Resourcen können für verschiedene Sprachen definiert werden, so gibt es AppleScript nicht nur in Englisch, sondern auch in Französisch und Japanisch. Eine deutsche Version blieb uns glücklicherweise erspart – die deutsche Sprache läßt sich halt nicht so einfach analytisch beschreiben ...

Das Erzeugen einer 'aete'-Resource ist leider nicht ganz einfach. Es gibt, soweit mir bekannt ist, vier Möglichkeiten:

1. mit dem 'aete'-HyperCard-Stack von den Entwickler-CDs. Leider mit einigen Fehlern und nicht für jedermann verfügbar. Dafür kann er automatisch Client und Server Sourcecode für Apple-Event-Routinen in Pascal und C erzeugen!
2. mit dem Rez-Compiler. Da dieser Compiler Symantec C++ und auch dem CodeWarrior beiliegt, dürfte dies die von den meisten bevorzugte Methode sein.
3. mit einer ersten Version eines speziellen brandneuen 'aete'-Editors für den ResEdit. Man kann ihn sicher in einigen Mailboxen finden. Leider ist er nicht ganz einfach zu bedienen.
4. mit dem kommerziellen Programm Resorcerer. Dies ist ein Ersatz für den bekannten Freeware Resource-Editor ResEdit. Er ist jedoch erheblich mächtiger und erlaubt das sehr einfache Editieren von 'aete'-Resourcen. Ich kann diesen Editor wirklich jedermann ans Herz legen. Nachteil: er ist vergleichsweise teuer: $256 ... Studenten können Rabatt erhalten.

recordable
Einen Schritt weiter gehen die sogenannten recordable-Programme. Diese Programme verschicken auch Apple Events für alle denkbaren Aktionen an sich selbst, z.B. bei der Anwahl von Menüeinträgen. Diese Events kann dann der Apple Script Editor aufzeichnen. Das Problem liegt hierbei im Wort „alle" – man muß wirklich alle Aktionen aufzeichenbar an sich selbst schikken – bereits bei mittelgroßen Programmen ein gewaltiger Aufwand!

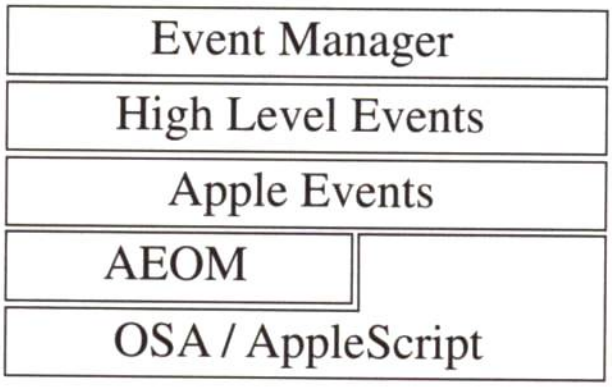
attachable
Ferner gibt es noch Programme, die attachable sind. Solche Programme ermöglichen das Einbinden von AppleScripts ins eigene Programm und das Ausführen solcher Scripte z.B. über einen Button oder einen Menüeintrag. Eine vergleichsweise einfache Funktionalität, die üblicherweise jedoch nur was bringt, wenn das eigene Programm auch scriptable ist.

AEOM

Noch eine Stufe weiter geht das **A**pple **E**vent **O**bjekt **M**odell – kurz AEOM. Es geht erheblich über die einfachen AppleScript-Befehle hinaus. Der Implementationsaufwand für große Applikationen ist jedoch normalerweise enorm! Hiermit kann man z.B. so etwas schreiben: *get the last character in the third word in the window named „Test"*

Dieser englische Ausdruck wird hier von rechts nach links abgearbeitet. Das Programm braucht dazu vier Routinen:
1. Ein Fenster mit dem Namen „Test" finden
2. Das dritte Wort in einem bestimmten Fenster finden
3. Den letzten Buchstaben in einem bestimmten Wort finden
4. Die Referenz auf ein gefundenes Objekt zurückgeben („get")

Die Apple-Event-Hierarchie



Ferner kann man auch noch Bedingungen formulieren, Objekte verändern, vergleichen usw. Kurzum: das Richtige, um sein Programm über einen langen Winter aufzupeppen ... Apple hat allerdings auch erst seit System 7.5 einen Finder geschaffen, der „scriptable" und „recordable" ist. Wer allerdings die PowerPlant Library vom CodeWarrior Compiler benutzt, hat es hier erheblich einfacher, da diese Library AEOM kräftig unterstützt!

Die MacOS-Funktionen

In diesem ersten Teil will ich zuerst den Aufbau der Apple Events erklären. Aus eigenen Erfahrungen muß ich gestehen, daß das NIM „Interapplication Communication" nicht gerade einfach zu verstehen ist. Zumal die Struktur im Buch nicht unbedingt toll gewählt wurde. Ich hoffe also, es hier besser zu machen. Allerdings werde ich kaum derart in die Tiefe gehen können, wie Apple im NIM, dazu fehlt mir hier natürlich der Platz.

Alle Apple-Event-Funktionen geben einen *OSErr* zurück, den man tunlichst auswerten sollte! Ich spare mir dies hier aus Platzgründen.

Descriptoren

Die kleinste Struktur in einem Apple Event sind die Descriptoren. Sie sind sehr einfach aufgebaut:

```
typedef struct {
    DescType descriptorType;
    Handle   dataHandle;
} AEDesc;
```

Jeder Descriptor besteht also aus einem Typ – 4-Byte groß – und einem Datenblock, der sich in einer Handle befindet. Mit der Funktion *GetHandleSize()* kann man die Größe des Datenblockes ermitteln. Ein besonderer Descriptor hat als descriptorType den Wert „typeNull" und als dataHandle NULL. Man nennt ihn – wer hätte es gedacht – Null-Descriptor. Man sollte übrigens nie auf den Inhalt des dataHandles direkt zugreifen, sondern immer nur über die MacOS-Funktionen. Wie das MacOS die Daten in diesem Handle ablegt, ist nämlich nicht dokumentiert – dies trifft insbesondere auf komplexe Listenstrukturen zu.

Einen Descriptor kann man mit der Funktion *AECreateDesc()* erzeugen:

```
AEDesc myDesc;
long   myLong = 42L;

  AECreateDesc(typeLongInteger, &myLong,
                  sizeof(myLong), &myDesc);
```

Diese Funktion setzt descriptorType in *myAEDesc* auf den Wert von typeLongInteger, legt ein dataHandle mit der Größe von *sizeof(myLong)* an und kopiert die Daten ab der Adresse *&myLong* in dieses Handle. Nichts Aufregendes also.

Descriptoren muß man wieder freigeben, dies geschieht mit *AEDisposeDesc()*:

```
AEDisposeDesc(&myDesc);
```

Diese Funktion kennt auch Null-Descriptoren und gibt in einem solchen Fall das – nicht existierende – dataHandle natürlich nicht frei. Man könnte mit *DisposeHandle()* auch selbst das dataHandle freigeben – dies geht in aller Regel auch gut – aber wenn der Descriptor ein Apple Event (kommt später) ist, dann sorgt *AEDisposeDesc()* auch für die Aktualisierung der Tabellen für das Verwalten der Reply-Events. Kurzum: man sollte für das Freigeben immer *AEDisposeDesc()* aufrufen.

AEDescList

Beim *kAEOpenDocuments* Apple Event wird eine Liste von Dokumenten übergeben. Diese wird in einer *AEDescList* abgelegt. Der Aufbau einer *AEDescList* entspricht einem *AEDesc*, jedoch werden in dem dataHandle alle Objekte der Liste einfach hintereinander abgelegt. Alle Objekte haben den gleichen Typ.

Mit der Funktion *AECreateList()* legt man eine leere Liste an:

```
AEDescList myDescList;
AECreateList(NULL, 0L, false, &myDescList);
```

Freigeben können wir diesen Descriptor wieder mit *AEDisposeDesc()*.

Mit den Funktionen *AEPutPtr()* und *AEPutDesc()* kann man Daten bzw. einen Descriptor an die Liste anhängen oder Einträge ersetzen. Mit folgender Zeile hängen wir ein Langwort mit dem Wert 42 an die Liste an:

```
long   myLong = 42L;
AEPutPtr(&myDescList, 0, typeLongInteger, &myLong,
                                  sizeof(myLong));
```

AERecord

Ein *AERecord* ist eine Liste, bei der die Einträge nicht nur über einen Index abgerufen werden können, sondern auch über ein Keyword.

Ebenfalls mit der Funktion *AECreateList()* legt man einen leeren Record an:

```
AERecord myDescRecord;
AECreateList(NULL, 0L, true, &myDescRecord);
```

Freigeben können wir diesen Descriptor wieder mit *AEDisposeDesc()*.

Mit den Funktionen *AEPutKeyPtr()* und *AEPutKeyDesc()* kann man Daten bzw. einen Descriptor an die Liste anhängen oder Einträge ersetzen. Mit folgender Zeile tragen wir ein Langwort mit dem Wert 42 unter dem Keyword 'MKEY' in die Liste ein:

```
long   myLong = 42L;
AEPutKeyPtr(&myDescList, 'MKEY', typeLongInteger,
                  &myLong, sizeof(myLong));
```

Apple Event

Ein Apple Event ist nichts anderes als ein *AERecord*, der bestimmte Descriptoren automatisch enthält. Ein Apple Event besteht aus zwei Descriptor-Gruppen: den Attributen, welche die nötigen Informationen enthalten um den Apple Event zu verschicken, und den Parametern, welche die eigentlichen Informationen enthalten. Dieser Aufruf wird schon etwas komplizierter:

```
AppleEvent  myEvent;
AEAddressDesc myDest;
AECreateAppleEvent('MYCL', 'MYID', &myDest,
  kAutoGenerateReturnID, kAnyTransactionID, &myEvent);
```

'MYCL' und *'MYID'* sind die Event-Klasse und die Event-ID, welche den Apple Event beim Empfänger identifizieren.

myDest ist hierbei ein Descriptor, welcher das Ziel des Apple Events beschreibt. Näheres dazu im nächsten Abschnitt.

kAutoGenerateReturnID legt automatisch eine eindeutige ID für den Reply-Event fest, den man als Antwort auf seinen Apple Event bekommt. Dies erlaubt das Zuordnen eines Replys zu einem früher abgeschickten Apple Event – man beachte, daß Antworten asynchron kommen, d.h., man kann drei Apple Events an einen Server schikken und bekommt erst später eine Antwort!

kAnyTransactionID sagt dem Apple Event Manager, daß dieser Apple Event zu keinem Block von Apple Events gehört, der zusammen ausgeführt werden muß.

Freigeben können wir diesen Descriptor wieder mit *AEDisposeDesc()*.

Target festlegen

Das Wichtigste an einem Apple Event ist die Festlegung, wer den Event bekommen soll. Der Empfänger wird mit einem *AEAddressDesc* beschrieben. Es gibt hierzu vier Möglichkeiten – die im NIM: Macintosh Toolbox Essentials genauer beschrieben sind – hier nur eine Kurzfassung:

1. typeApplSignature
Hiermit kann man als Empfänger den File-Creator einer Applikation angeben. Das Programm muß auf dem gleichen Rechner bereits laufen.

```
OSType theSignature = 'MauS';
AECreateDesc(typeApplSignature, &theSignature,
        sizeof(theSignature), &targetDesc);
```

2. typeProcessSerialNumber
Hiermit kann man eine *ProcessSerialNumber* (PSN) angeben. Dies setzt voraus, daß der Prozeß auf dem gleichen Rechner läuft. Ferner ist es die empfohlene Methode, Apple Events an sich selbst zu schicken: man gibt *kCurrentProcess* als PSN an. Man schickt Events an sich selbst, um recordable zu sein. Schickt man den Apple Event an *kCurrentProcess*, so wird der Event Dispatcher ignoriert, um statt dessen direkt in die Callback-Routine zu springen. Somit kostet das Schicken eines Events an sich selbst kaum Rechenzeit.

3. typeSessionID
Die *Session-ID* benutzt man immer dann, wenn man auf einen High Level Event antworten will. Dies ist die einfachste und schnellste Methode, um einem Absender Daten zu schicken. Dies funktioniert auch über ein Netzwerk.

4. typeTargetID
Die *targetID* beschreibt einen Prozeß auch über Rechnergrenzen hinweg, also im Netzwerk auf fremden Rechnern. Man ermittelt die *targetID* üblicherweise mit der Funktion *PPCBrowser()*. Eine sehr praktische Methode, um bei einem Netzwerkspiel Mitspieler zu suchen.

Wer Apple Events über das Netzwerk empfangen will, muß dies in der SIZE-Resource explizit erlauben!

Apple Event verschicken

Zu guter Letzt können wir unseren Apple Event nun abschicken:

```
AppleEvent  myEvent;
AppleEvent  myReply;
AEIdleUPP   myIdleUPP = NewAEIdleProc(myIdleFunc);
 AESend(&myEvent, &myReply, kAEWaitReply +
kAENeverInteract, kAENormalPriority, 120, myIdleUPP, NIL);
 DisposeRoutineDescriptor(myIdleUPP);
```

myEvent ist unser vorher zusammengestellter Apple Event. *myReply* wird, wenn kein Fehler auftritt, die Antwort auf unseren Event enthalten.

kAEWaitReply besagt, daß wir mit der Ausführung von unserem Programm warten wollen, bis eine Antwort anliegt. Dies setzt voraus, daß wir eine Idle-Function zur Verfügung stellen. Bei mir heißt sie *myIdleFunc*. Diese entspricht der normalen Event-Auswertung, jedoch bekommt man keine Tastendrücke, Mausklicks etc. Redraw von Fenstern bzw. einen Update-Event muß man dort aber ausführen. Auch kann man Null-Events für das Blinken eines Cursors oder die Animation des Mauszeigers nutzen.

kAENeverInteract besagt, daß der Empfänger unseres Events niemals den User nach irgendwelchen Dingen fragen soll. Muß der Empfänger irgendwas vom User wissen, so gibt es einen Fehler. Die Interaktion ist gerade bei einem Server, der zwei Räume weiter steht, natürlich nicht immer erwünscht.

kAENormalPriority bewirkt eine normale Abarbeitung des Apple Events. Mit *kAEHighPriority* würde der Apple Event vor allen anderen noch ausstehenden Events bearbeitet werden.

Wir setzen einen Timeout für den Reply von 120 Ticks, dies sind 2 Sekunden.

Der letzte Parameter ist *NIL*, wir haben keine spezielle Filter-Funktion für Apple Events.

Als Besonderheit ist zu beachten, daß mit *NewAEIdleProc()* ein Universal Proc Ptr definiert wird, der nach dem Event mit *DisposeRoutineDescriptor()* freigegeben wird. Dies ist nötig, damit unser Programm auch auf einem PowerMac funktioniert. Es funktioniert natürlich auch noch auf 68k-Macs. Weiteres steht im NIM: PowerPC System Software.

Empfangen von Apple Events

Alle Apple Events, die man kennt, muß man beim Programmstart mit der Funktion *AEInstallEventHandler()* anmelden. Ruft man in seiner Event-Auswertung bei Erhalt eines Apple Events dann die Funktion *AEProcessAppleEvent()* auf, sorgt das MacOS automatisch für den Aufruf der richtigen Routine. Einen eigenen Dispatcher kann man sich also sparen.

Die Callback-Routine hat folgenden Prototyp:

```
OSErr myAEHandler(AppleEvent theAppleEvent,
            AppleEvent reply, long refCon);
```

Man bekommt also den Apple Event übergeben. Aus diesem kann man sich z.B. mit *AEGetParamDesc()* die gewünschten Parameter herausziehen.

In den *reply* kann man seine Antwort ablegen, z.B. einen String zurückgeben.

refCon enthält den Wert, den man bei *AEInstallEventHandler()* angegeben hat. Somit kann man die gleiche Callback-Routine für verschiedene Events nutzen, man kann sie anhand des *refCon*-Wertes unterscheiden.

Als Rückgabewert gibt man einen *OSErr* zurück. Man sollte stets *noErr* zurückgeben, wenn man den Apple Event erfolgreich abarbeiten konnte! Wenn der User eine Funktion abbricht, gibt man –128 (dies ist der Standardfehlercode für einen Abbruch) zurück.

Der Beispielecode

Ich gebe diesmal nur einige Programmfragmente an, da ein komplettes Listing deutlich zu lang sein würde. Man kann diese Fragmente problemlos in eigene Programme einbauen oder auch in das Beispielprogramm aus der MacOPEN Mai 95.

Und nun? So, ein Anfang ist gemacht. Ich hoffe, der erste Teil unseres Einstieges in die Apple Events reizt zu mehr. Im nächsten Teil werde ich dann ein ausführlicheres Beispiel von zwei Programmen, die mit Apple Events Daten austauschen, liefern. Diese Programme werden dann auch scriptable und recordable sein.

Ferner werde ich ein paar Worte bezüglich des Debuggens von Programmen mit Apple Events verlieren.

Bis dahin empfehle ich schonmal die Lektüre des NIM: Interapplication Communication ...

MFR

Literatur:

Inside Macintosh: Interapplication Communication (von Apple Computer, Addison-Wesley), ISBN 0-201-62200-9

Inside Macintosh: PowerPC System Software (von Apple Computer, Addison-Wesley), ISBN 0-201-40727-2

Inside Macintosh: Macintosh Toolbox Essentials (von Apple Computer, Addison-Wesley), ISBN 0-201-63243-8


```
 1: /***
 2:  *(c) 1995 MAXON Computer
 3:  *Autor: Markus Fritze
 4:  *Testen, ob Apple Events vorhanden sind.
 5:  *Muß beim Programmstart getestet werden.
 6:  *
 7:  *return code != noErr, wenn es Fehler gibt
 8:  ***/
 9: OSErr AEInit(void)
10:
11: {
12: OSErr myErr;
13: long  attr;
14:
15:  myErr = Gestalt(gestaltAppleEventsAttr, &attr);
16:  if(myErr != noErr)
17:   return myErr;
18:  // Apple Events vorhanden?
19:  if(!(attr & (1L << gestaltAppleEventsPresent)))
20:   return -1; // Nein => raus
21:
22:  return noErr;
23: }
24:
25: /***
26:  *dies gehÖrt in die Event Auswertung rein:
27:
28:  *Erhält das Programm einen Apple Event, so
29:  *leitet man den Event direkt an den Apple
30:  *Event Manager weiter, der dann unsere
31:  *Callback-Routinen aufruft.
32:  ***/
33:  switch(theEvent.what) {
34:  case kHighLevelEvent: AEProcessAppleEvent(&theEvent);
35:                        break;
36:  }
37:
38: /***
39:  *Nachdem wir alle Parameter eines Apple
40:  *Events abgeholt haben, sollten wir - laut
41:  *Apple - testen, ob noch unbearbeitete
42:  *Parameter übrig geblieben sind. Wenn dies
43:  *der Fall wère, brechen wir die Event
44:  *Bearbeitung mit einem Fehler ab!
45:
46:  *Der Test ist recht einfach: falls wir nicht
47:  *alle - nicht optionalen! - Parameter
48:  *abgeholt haben, existiert ein besonderes
49:  *Attribut im AppleEvent:
50:  *"keyMissedKeywordAttr", welches besagt: es
51:  *gibt noch mehr! Wenn dieses Keyword fehlt,
52:  *dann haben wir alles gelesen.
53:  ***/
54: OSErr CheckMissedParm(AppleEvent theAppleEvent)
55: {
56: OSErr    myErr;
57: DescType returnedType;
58: Size     size;
59:
60:  myErr = AEGetAttributePtr(theAppleEvent,
      keyMissedKeywordAttr, typeWildCard, &returnedType, NIL,
       0, &size);
61:  if(myErr == errAEDescNotFound)
62:   return noErr;
63:  return errAEParamMissed;
64: }
65:
66: /***
67:  *So sieht z.B. ein kAEOpenDocuments Apple
68:  *Event aus.
69:
70:  *Den reply brauchen wir nicht zu setzen - es
71:  *wird keine Antwort erwartet. refCon ist bei
72:  *uns ebenfalls unbenutzt.
73:  ***/
74: OSErr myOpenDoc(AppleEvent theAppleEvent, AppleEvent
    reply, long refCon)
75: {
76: OSErr      myErr;
77: AEDescList docList;
78:
79:  // Liste von Descriptoren aus dem direkten Parameter
     erzeugen
80:  myErr = AEGetParamDesc(theAppleEvent, keyDirectObject,
     typeAEList, docList);
81:  if(myErr != noErr) return myErr;
82:
83:  myErr = CheckMissedParm(theAppleEvent);
84:  if(myErr != noErr) goto raus;
85:
86:  // Anzahl der Elemente der Liste ermitteln
87:  short count;
88:  myErr = AECountItems(docList, &count);
89:  if(myErr != noErr) goto raus;
90:
91:  // alle Einträge der Liste abarbeiten
92:  for(int index=1; index<count; index++) {
93:
94:   // n. Parameter als FSSpec aus der Liste ziehen
95:   // evtl. Typwandlungen macht das MacOS automatisch
96:   AEKeyword keyw;
97:   DecsType  retType;
98:   FSSpec    fsp;
99:   Size      size;
100:  myErr = AEGetNthPtr(docList, index, typeFSS, &keyw,
      &retType, &fsp, sizeof(fsp), &size);
101:  if(myErr != noErr)
102:   break;
103:
104:  // Dokument öffnen
105:  OpenDocument(&fsp);
106: }
107:
108: // im Fehlerfall müssen wir trotzdem unsere Liste
109: // freigeben!
110: raus:
111:  // unsere Liste freigeben
112:  AEDisposeDesc(docList);
113:  return myErr;
114: }
```