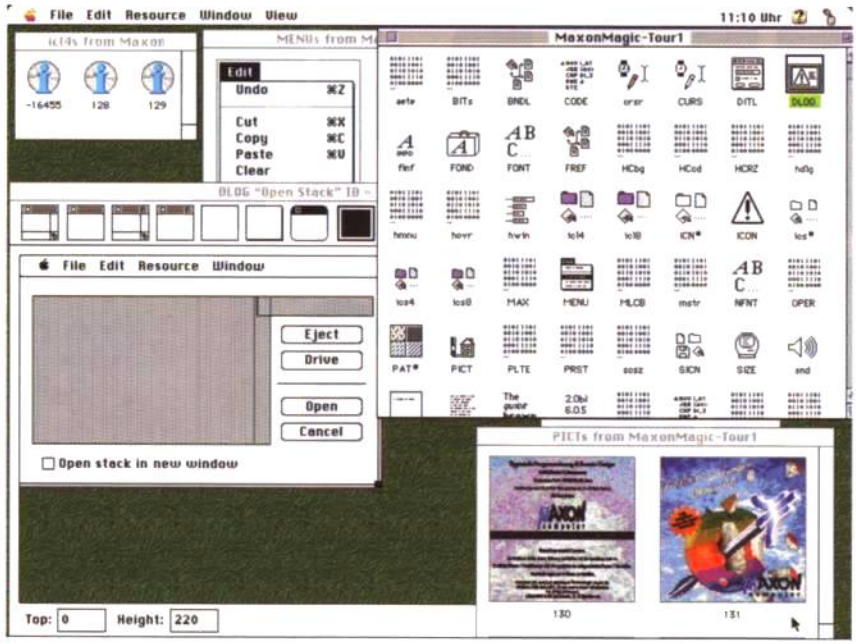

MacOPEN

Software

Hardware

Grundlagen

Programmieren auf dem Mac

Das MacOS

Heute will ich mich mal auf ein ganz heißes Pflaster wagen: einen Systemvergleich von TOS und MacOS. Allerdings nicht, um einen Sieger zu küren, sondern um zu zeigen, was beim Mac anders ist als beim ATARI. Nun ja, ein echter Systemvergleich ist es nicht. Ich werde vielmehr die speziellen Eigenheiten des MacOS beschreiben – immer wieder mit Blick auf den ATARI.

Ein Schichtenmodell, wie BIOS bzw. XBIOS, GEMDOS, VDI und AES, gibt es beim Mac nicht. Das MacOS stammt zwar aus der gleichen Ära der Computerzeit, trotzdem haben die damaligen Entwickler keine Anleihen bei CP/M oder UNIX gemacht – egal ob positiv oder negativ.

Das MacOS besteht aus einer schier unüberschaubaren Anzahl von mehr oder weniger unabhängigen Librarys, sogenannten Managern. Es gibt einen für die Dateiverwaltung, einen für die Fensterverwaltung, mehrere für die Grafik usw. Alle werden in den Inside-Macintosh-Bänden von Addison Wesley beschrieben. Da aber alle paar Monate ein paar dazukommen, hinkt man ständig hinterher. Glücklicherweise braucht man nicht alle neuen Betriebssystemfunktionen. Viele braucht man nicht einmal zu kennen.

Einige in diesem Text beschriebenen Features gibt es erst seit System 7. Sie sind auf älteren Versionen nicht oder nur in Verbindung mit bestimmten Systemerweiterungen vorhanden. Spectre und einige wenige Mac-User können sie also nicht benutzen.

Ein ordentlicher Programmierer fragt natürlich vor dem Benutzen solcher Funktionen mit der *Gestalt()*-Funktion ab, ob die jeweiligen Funktionen überhaupt vorhanden sind. *Gestalt()* ist so etwas wie der Cookie-Jar beim ATARI, jedoch ohne daß der Benutzer in einer Liste suchen und patchen muß, sondern ordentlich mit den Funktionen der Gestalt-Managers. Im aktuellen MacOS 7.5.1 gibt es weit über 100 Einträge in der Gestalt-Liste. Man kann sie sich mit einigen PD-Programmen ansehen.

Das MacOS und Pascal-Strings

Viele MacOS-Betriebssystemfunktionen erwarten als Übergabeparameter Pascal-Strings. Unter TOS sind es im Gegensatz dazu stets C-Strings.

Und weil dem unter dem MacOS so ist, haben alle Mac-C-Compiler eine „nicht-ANSI-Erweiterung", die es ermöglicht, daß man statische Strings direkt als Pascal-Strings definieren kann. Dazu schreibt man als erstes Zeichen eines Pascal-Strings lediglich '\p'. Hier ein Beispiel:

```
C:      "abc" = 'a' 'b' 'c' '\0'
Pascal: "\pabc" = '\3' 'a' 'b' 'c'
```

Wie man sieht: C-Strings sind durch ein Null-Byte abgeschlossen, Pascal-Strings ist ein Längen-Byte vorangestellt.

Ferner gibt es noch zwei häufig benutzte Hilfsfunktionen: *CtoPstr()*, welche einen C-String in einen Pascal-String wandelt; *PtoCstr()*, welche einen Pascal-String in einen C-String wandelt.

Achtung!

Bei beiden Funktionen gibt es Zweierlei zu beachten: ein Pascal-String darf nur maximal 255 Zeichen lang sein. Versucht man einen längeren C-String zu wandeln, so wird er auf 255 Zeichen verkürzt. Ein Pascal-String kann Null-Bytes enthalten. Wandelt man ihn in einen C-String, wird der C-String an dieser Stelle abgebrochen.

Das zweite ist, daß Zeichen in Pascal-Strings stets vom Typ „unsigned char" sind, wohingegen Zeichen in C-Strings normalerweise vom Typ „char" sind. Dies macht es bei strengen Prototypen in ANSI-C leider nötig, häufig zu „typecasten".

Aber warum heute noch Pascal-Strings? Zum einen natürlich aus historischen Gründen, schließlich wurden die ersten Versionen des MacOS in Pascal programmiert, und viele der älteren MacOS-Funktionen braucht man auch heute noch.

Der File Manager

Besonders wichtig sind Pascal-Strings auch heute noch bei Dateinamen. Die-

se dürfen beim MacOS nämlich nicht nur bis zu 31 Zeichen lang sein – sie dürfen mit Ausnahme des Doppelpunktes ':' auch alle anderen 255 Zeichen enthalten. Auch das Null-Byte!

Aber wer setzt schon Null-Bytes in Dateinamen? Nun, einige! Systemerweiterungen und Kontrollfelder innerhalb des Systemordners werden nämlich in alphabetischer Reihenfolge geladen. Virendetektoren und Startup-Manager können durch ein oder mehrere Null-Bytes am Anfang des Dateinamens sicherstellen, daß sie ganz am Anfang geladen werden.

Als Programmierer hat man so gut wie nie mit Pfadnamen zu tun. Das ist eine wirklich angenehme Tatsache, wenn man bedenkt, wieviel Arbeit man mit dem richtigem Aufbereiten von Datei- und Pfadnamen unter TOS hat. Die einzelnen Ordnernamen werden beim Mac übrigens nicht durch '\' getrennt, sondern durch ':'.

Seit System 7 wird nur noch mit einer Struktur namens *FSSpec* gearbeitet. Ein *FSSpec* ist nichts anderes als eine Struktur, die den Dateinamen, den Ordner, in dem die Datei liegt, und das Laufwerk auf dem sich der Ordner befindet, beschreibt.

```
typedef struct FSSpec
{
   short vRefNum;
   long  parID;
   Str63 name;
} FSSpec,*FSSpecPtr;
```

name ist nichts anderes als der bekannte Pascal-String. Der Ordner ist aber nicht durch seinen Namen, sondern nur durch eine *parID* beschrieben! Die *parID* ist, solange der Ordner auf einem Laufwerk existiert, eindeutig. Man kann also problemlos einen Ordner mit offenen Dateien im Finder verschieben. Die *vRefNum* ist die Kennung des Laufwerkes, sie wird bei jedem Mounten, also wenn das Laufwerk auf dem Desktop erscheint, neu zugeteilt.

Die MacOS-Funktionen *StandardGetFile()* oder *StandardPutFile()* entsprechen der Fileselectorbox des ATARIs. Sie sind jedoch in weitesten Grenzen frei programmierbar. Eigene Dateifilter oder ein eigenes Aussehen sind nur zwei Möglichkeiten davon. Diese Funktionen ergeben bzw. erwarten praktischerweise einen *FSSpec*.

Alias

Um sich die Position einer Datei oder eines Ordners längere Zeit zu merken, z.B. in den Voreinstellungen, legt man am besten ein Alias auf diese Datei an. Ein Alias ist eine Referenz auf eine Datei, die erheblich stabiler als das Merken von Dateinamen etc. ist. So kann ein Alias eine Datei auch dann wiederfinden, wenn ein Benutzer die Datei oder das Laufwerk umbenannt hat. Selbst wenn die Datei verschoben wird, kann sie noch wiedergefunden werden.

Data- und Resource-Fork

Ein besonderes Highlight des File-Managers ist seine Fähigkeit, unter einem Namen zwei Dateien zu verwalten. Für den Anwender verhalten sich diese zwei Dateien jedoch stets wie eine Datei, z.B. beim Kopieren. Man nennt die eine Datei *Data-Fork* und die andere *Resource-Fork*. Data-Fork entspricht einer normalen ATARI-Datei, kann also Text, Grafik oder was auch immer enthalten. Resource-Fork hat ein bestimmtes Format, das am ehesten mit einer kleinen Datenbank zu vergleichen ist, die über den sogenannten Resource-Manager angesprochen wird. Dazu mehr im nächsten Abschnitt.

Die MacOS Funktionen des File-Managers sind ziemlich reichhaltig. Es gibt wohl über hundert. Glücklicherweise braucht man normalerweise nur ein paar, nämlich die Funktionen, die man vom GEMDOS her auch kennt: *FSOpenDF()*, *FSRead()*, *FSWrite()*, *FSClose()*.

Neben den üblichen Dateiattributen wie *invisible* o.ä. werden bei Mac-Dateien auch noch die Pixel-Koordinaten des Datei-Icons im Fenster bei den Dateien direkt mit abgelegt. Dies erklärt, warum der Finder so problemlos die Dateien in Fenstern verschieben kann und die Position auch unter den verrücktesten Umständen noch stimmt – es sei denn, der Mac ist abgestürzt, bevor er seine Änderungen wegschreiben konnte.

Type und Creator

Zu guter Letzt gibt es noch zwei ganz wichtige Informationen, die jede Datei hat: *Filetype* und *Creator* – jeweils 4 Byte lang. Der Filetype einer Datei gibt an, was diese Datei üblicherweise in der Data-Fork enthält. Ist es z.B. 'TEXT', sollten in der Data-Fork nur Textzeilen (durch CR getrennt) stehen. Zeilen werden beim Mac nämlich nur durch ein einzelnes CR und nicht durch CR und LF getrennt wie beim ATARI oder unter DOS. Alle Programme, die von sich behaupten, TEXT-Dateien lesen zu können, können diese Datei per Drag&Drop im Finder öffnen.

Der Filetype 'APPL' kennzeichnet normale Mac-Programme, 'INIT' steht für einfache Systemerweiterungen. Es gibt noch viele weitere vordefinierte Typen. Einen Filetype für eigene Dateiformate kann man sich selbst ausdenken – er sollte aber möglichst eindeutig sein! Nur Kleinbuchstaben sind verboten, die hat sich Apple reserviert.

Der Creator muß für jedes Programm auf der Welt unbedingt eindeutig sein! Der Creator ist die Kennung für jedes einzelne Programm und seine Dateien, wie z.B. 'R*ch' für den BBEdit, mit dem ich gerade diesen Text schreibe, oder 'MauS' für die Mausefalle für den MausTausch. Ein Creator für ein eigenes Programm muß bei Apple beantragt werden! Glücklicherweise ist dies problemlos und vor allen Dingen kostenlos möglich.

Macht man im Finder einen Doppelklick auf eine TEXT-Datei, so wird zuerst versucht, das passende Programm zu dieser Datei zu finden. Falls es nicht gefunden wird, präsentiert einem der Finder eine Auswahl an Programmen, die diese Datei alternativ öffnen können (*EasyOpen* muß dazu installiert sein!).

Der File-Manager wird übrigens auch zur Ansteuerung von Device-Treibern benutzt. So kann man durch Öffnen der Datei „.AOut" Daten auf dem Modem-Port ausgeben. „.Sony" ist der Diskettentreiber usw. Device-Namen fangen immer mit einem Punkt an, man sollte also einen Punkt am Anfang eines Dateinamens möglichst meiden.

Da Devices natürlich im Interrupt angesprochen werden können, mußte der File-Manager reentrant sein – was

er auch ist. Man kann problemlos in einem Timer-Interrupt eine Datei erzeugen und Daten dort hineinschreiben.

Der Resource-Manager

Der Resource-Manager verwaltet die sogenannte Resource-Fork einer Datei. Alle Programme haben eine Resource-Fork. Alle Dokumente sollten eine haben, nötig ist sie dort allerdings nicht. Die Resource-Fork eines Programms enthält neben dem 68K-Programmcode auch noch das, was man beim ATARI in einer RSC-Datei lagert: Menüs, Dialogboxen, Icons und vieles mehr.

Die Resource-Fork ist dabei wie eine kleine Datenbank aufgebaut: es gibt quasi beliebige viele Kategorien, die man per 4-Zeichen-Kürzel abfragen kann: 'DLOG' für Dialogboxen, 'STR ' für Strings, 'STR#' für String-Listen, 'PICT' für Bilder usw. Mit der Funktion *GetResource('PICT', 128)* kann man das Bild 128 ins RAM laden – um es z.B. mit *DrawPicture()* zu zeichnen.

Resourcen kann man mit dem frei verfügbaren *ResEdit* von Apple editieren. Dieser Editor kann Hunderte von Resource-Typen darstellen und editieren. Man kann sich sogar eigene Editoren für selbst erdachte Resource-Typen schreiben! Wem das zu aufwendig ist, der kann mit Templates einfache Eingabemasken für seine eigenen Typen erzeugen. Den ResEdit braucht wirklich jeder Mac-Programmierer!

Der Resource-Manager bietet alle Funktionen, um diese Datenbank zu durchsuchen, zu ergänzen oder sonstwie zu pflegen. Durch die konsequente Ausnutzung der Auslagerung aller landesspezifischen Dinge, wie Texte, Regeln etc., kann man seine Programme später sehr leicht in andere Sprachen übertragen. Am besten, man schaut sich mit dem ResEdit mal ein paar Programme an, um das Prinzip zu vertiefen.

Der Memory-Manager

Die Speicherverwaltung vom Mac ist im Vergleich zum TOS eine Welt für sich. Sie ist sehr schnell, stabil und hat auch keine Probleme bei weit mehr als 10000 allozierten Speicherblöken in einem Programm. Sie gehört zu den Teilen vom MacOS, der sich seit über 10 Jahren aus Sicht des Programmieres so gut wie nicht verändert hat. Sie funktioniert vom Prinzip her aber anders als die auf allen anderen Rechnern. Zuerst einmal gibt es einen eigenen Heap für jedes aktive Programm; wird es beendet, wird der Application-Heap verworfen. Ferner gibt es noch einen System-Heap, in dem Speicherblöcke alloziert werden, die auch nach Beenden eines Programms noch erhalten bleiben sollen. Üblicherweise liegen dort auch die Systemerweiterungen und Kontrollfelder.

Man arbeitet auf dem Mac auch nicht mit Pointern auf Speicherblöken, sondern mit Handles. Handles sind Pointer auf Masterpointer, die auf den eigentlichen Block zeigen. Eine Dereferenzierung mehr – klingt umständlich? Ist es auch, aber nur wenig. Man gewinnt jedoch einige Vorteile: Da ein Programm nie die absolute Adresse eines Speicherblockes kennt, kann die Speicherverwaltung diesen Block bei Bedarf frei verschieben. Somit kann sie Speicher zusammenziehen, um Platz für einen größeren Block zu schaffen. Eine Fragmentierung des Speichers kann im Idealfall nicht mehr auftreten. Bei Bedarf kann man das Verschieben der Blöcke allerdings verhindern:

```
void FillBlockWithRandom(Handle h)
{
   HLock(h);
   Ptr p = *h;
   long size = GetHandleSize(h);
   while(size-->0)
      *p++ = Random();
   HUnlock(h);
}
```

In diesem Beispiel ist das Locken des Blockes allerdings sogar unnötig, denn es werden keine Betriebssystemfunktionen aufgerufen, die Speicher verschieben können. Nun ja, was tut man nicht alles für ein gutes Beispiel ... Leider wird in anderen Situationen sehr häufig vergessen, Speicherblöke zu locken. Das Ergebnis sind sehr sporadische Abstürze.

Ferner kann man Speicherblöcke als „purgeable" definieren. Wenn der Speicher nicht mehr ausreicht, werden diese Blöcke automatisch freigegeben. Ideal für Buffer und Caches! Vor der Benutzung muß man sie dann natürlich wiederbeleben. Man erkennt sie daran, daß der Masterpointer undefiniert, also **h == nil* ist.

Mit den Funktionen *NewPtr()* und *DisposePtr()* gibt es allerdings auch noch zwei Funktionen, um eine Pointer-basierende Speicherverwaltung à la ATARI unter dem MacOS zu nutzen. Für eine einfache Portierung nicht unpraktisch, aber in echten Mac-Programmen sollte man sie meiden.

QuickDraw

Das klassische QuickDraw stammt ebenfalls noch aus dem Ur-Mac. Es wurde vor einigen Jahren zuerst durch Color QuickDraw (bis zu 256 Farben) und dann durch 32-Bit Color QuickDraw (True-Color) ersetzt. Die Programmiererseite ist jedoch relativ unverändert geblieben.

QuickDraw ist im Vergleich zum VDI in vielen Punkten etwas schwach bestückt: Pfeile oder ähnliches gibt es z.B. nicht. Dafür hat es einige sehr mächtige Funktionen, die ich mir auf dem ATARI immer gewünscht habe: neben den üblichen Polygonen gibt es Regionen und ein erheblich mächtigeres Clipping.

Regionen ...

... sind eine Obermenge von Polygonen: sie können beliebige Flächenformen umfassen. Selbst Löcher sind kein Problem. Diese Regionen werden beim Mac sehr gerne für das Clipping eingesetzt. Anstatt mit Rechtecklisten rumzuspielen, bekommt man einfach eine Region übergeben, die beschreibt, welcher Teil des Windows neu gezeichnet werden muß. Wer einmal das Spiel *SimEarth* auf dem Mac gesehen hat, weiß, daß es auch problemlos runde Fenster gibt: hier helfen einem rechteckige Clipping-Bereiche nicht sonderlich weiter ... Nebenbei sind Regionen nicht einmal besonders langsam.

Die diversen Clipping-Möglichkeiten des MacOS sind sehr viel besser ausgeprägt als unter dem VDI. So ist es gar nicht möglich, außerhalb eines Windows zu zeichnen – wenn man es nicht unbedingt will, es wird stets auf

mehreren Ebenen geclippt (eigenes Programm, Window, Bildschirm).

QuickDraw arbeitet mit sogenannten *GrafPorts*. Ein *GrafPort* ist nichts anderes als eine Struktur, die alle Informationen über einen Ausgabebereich enthält. Jedes Window hat z.B. seinen eigenen *Grafport* mit eigenem Koordinatensystem, eigenem Clipping, usw. Ein *Grafport* ist also etwa so etwas wie eine *Workstation* unter dem VDI. Um etwas in einem Window auszugeben, muß man den *Grafport* setzen:


```
void PrintHallo(GrafPtr p)
{
  GrafPtr savePort;
  GetPort(&savePort);
  SetPort(p);
  MoveTo(30,30);
  DrawString("\pHallo Welt!");
  SetPort(savePort);
}
```

Ich habe hier gleich noch einmal gezeigt, wie man den alten *Grafport* rettet und später zurücksetzt, um den Kontext nicht zu verändern. Ein falsch gesetzter *Grafport* kann einem stundenlange Fehlersuche bescheren ...

Der Event-Manager

Der Event-Manager entspricht vom Prinzip dem Event-Handling des AES. Allerdings gibt es beim Mac eben kein darunter liegendes VDI, GEMDOS oder BIOS. Tastendrücke, Maustastendrükke bekommt man halt nur per Events.

Ferner gibt es zwei Events für die Fensterverwaltung: ein *Update Event*, wenn ein Fenster neu gezeichnet werden muß, ein *Activate Event*, wenn ein Fenster getoppt oder untoppt wird.

Zudem gibt es *High-Level-Events*, die sogar über das Netzwerk verschickt werden können. Außerdem werden die Apple-Events in High-Level-Events eingebettet, eine Basis für *AppleScript*, die Script-Sprache für die Automatisierung von Programmen.

Der Window-Manager

Im Vergleich zum AES ist das Macintosh-Fenster-Handling sehr rudimentär. Man muß sehr viel selbst programmieren! Dazu gehört auch das richtige Toppen von Fenstern, wenn sie angeklickt wurden. Dies klingt gewaltig aufwendig, ist es aber nicht. Einmal programmiert, hat man eine erheblich größere Flexibilität als unter TOS. So kann man beim Klick in den Fenstertitel, wenn die Command-Taste gedrückt wird, ein Pop-up-Menü hervorzaubern, um bestimmte Infos anzeigen zu können.

Ferner kann man beim Mac problemlos die *Window Definition Function* 'WDEF' komplett ersetzen. Somit kann man selbst runde oder sonstige Fensterformen programmieren. Das restliche Programm ändert sich durch eine neue 'WDEF' nicht.

Der Dialog-Manager

Ein Dialog ist beim Mac nichts anderes als ein Fenster mit ein paar Erweiterungen. Der Mac kennt als normale Kontrollelemente Buttons, Radio-Buttons, Checkboxes, Bilder und Useritems. Letztere kann und muß man durch eigene Funktionen zur Laufzeit ersetzen. Eigene Typen kann man sich jederzeit hinzuprogrammieren, per 'CDEF' – *Control Definition Function*.

Leider ist es mit der Dialog-Verwaltung beim Mac nicht besonders weit her: keine Gruppen von Radio-Buttons, nichts. Man muß alles selbst programmieren. Sobald eine Checkbox angeklickt wurde, wird man sie üblicherweise toggeln wollen: selbst machen! Mit etwas Routine ist das aber relativ einfach. Wenn Interesse besteht, kann ich ja mal eine Library für fertige Dialogboxen als Listing veröffentlichen.

Der Menü-Manager

Bei den Menüs ist es ähnlich wie bei den Fenstern: man muß fast alles selbst machen, hat dadurch aber eine große Flexibilität. Auch hier gibt es eine Menü-Definitionsfunktion 'MDEF', die man frei verändern kann.

Das Beispielprogramm

Ich bitte, das Beispielprogramm nicht allzu ernstzunehmen. Es tut absolut nichts Sinnvolles. Man kann beliebig viele Fenster öffnen, diese verschieben, vergrößern etc. Ich will damit auch lediglich die grundlegende Struktur eines MacOS-Programms aufzeigen.

Interessant ist allerdings die Erkennung, ob ein Mehrfachklick vorliegt. Der Mehrfachklick muß nicht nur in einem bestimmten Zeitrahmen erkannt werden, der Mauszeiger darf sich in dieser Zeit auch nicht zu weit wegbewegt haben. Die Art der Erkennung sorgt zudem dafür, daß, bevor ein Doppelklick erkannt wird, immer erst ein Einfachklick erkannt wird! Dies ist auf dem Mac auch so üblich: erster Klick: Cursor setzen; zweiter Klick: Wort selektieren; dritter Klick: Zeile selektieren.

Die Erzeugung eines Programms aus diesem Listing ist leider nicht ganz trivial – ich verkneife mir mal den Begriff „tierisch umständlich" und beschreibe im folgenden die Vorgehensweise mit dem *CodeWarrior C++ Compiler*.

1. Eine neue Projektdatei mit dem Namen „STC Demo.µ" anlegen. Den Compiler auf C++ schalten.

2. Das Listing sowie die Libraries „CPlusPlus.lib" und „MacOS.lib" dem Projekt hinzufügen. Die Libraries findet man im „Libraries *f*"-Verzeichnis des Compilers.

3. Das Projekt sollte sich nun übersetzen lassen. Aber es funktioniert noch nicht ... die Resourcen (für die Menüs und den About-Alert) fehlen noch.

4. „STC Demo.r" mit Hilfe des Toolservers (der richtig und vollständig installiert sein muß) und dem Rez übersetzen. Folgende Commandline sollte dies tun:

```
Rez -o 'STC Demo.µ.rsrc' -t rsrc -c
RSED 'STC Demo.r' -s 'PowerMac
HD:Development:Metrowerks C/C++
f:ToolServer 1.1.1:Interfaces:RIncludes:'
```

Ein ziemlicher Gigant ... Den Pfad für die *RInclude*-Dateien muß man übrigens anpassen.

5. Nun sollte man eine Datei „STC Demo.µ.rsrc" haben, die man zur Projektdatei kopiert – oder dieser einfach hinzufügt.

6. Ein Make (Command-M) sollte nun zum gewünschten Ergebnis und einem lauffähigen Programm führen.

Keine Panik, normalerweise hat man mit diesen *.r-Dateien, dem Toolserver und dem Rez nichts zu tun. Man wird seine Resourcen immer mit dem

ResEdit entwerfen und editieren. Leider kann man solche binären Daten schlecht in einer Zeitschrift abdrucken, deswegen der vergleichsweise umständliche Weg.

Eine bereits kompilierte *.rsrc-Datei inkl. Projektdatei kann man z.B. in der MAUS HH2 (040-52682040) finden oder von mir online abfordern (mfritze@mail.hh.provi.de) – sie sind nur wenige Bytes groß. (Hinweis der Red.: alle notwendigen Dateien zu diesem Artikel finden Sie auch auf der Megadisk zu diesem Heft. Um die Dateien von der Megadisk auf Ihre Festplatte kopieren und entpacken zu können, brauchen Sie ein installiertes *PC-Exchange*. Es wird inzwischen jedem Mac beigelegt). Wer sofort loslegen will, kann auch versuchen, mit dem ResEdit eine MBAR und die drei MENU-Resourcen zu erzeugen – es ist nicht schwierig. Der ALRT für die About-Box ist ebenfalls unkritisch. Viel Spaß!

MFR

Literaturhinweise:

Inside Macintosh:
Macintosh Toolbox Essentials (Event Manager, Menu Manager, Window Manager, Dialog Manager)
ISBN 0-201-63243-8

Inside Macintosh: Imaging with QuickDraw
ISBN 0-201-63242-X

Inside Macintosh: Files
ISBN 0-201-63244-6

Inside Macintosh:
More Macintosh Toolbox (Resource Manager, u.v.a.m.)
ISBN 0-201-63299-3

Inside Macintosh: Memory
ISBN 0-201-63240-3

Inside Macintosh CD-ROM (enthält alle fast 30 NIM-Bände)
ISBN 0-201-40674-8

Macintosh C Programming Primer Volume I ISBN 0-201-15662-8

ResEdit Complete
ISBN 0-201-55075-X

```
/***

 * STC Demo.cp
 * Ein winziges Demoprogramm um die grundlegenden
 * Features des MacOS aufzuzeigen.
 * (c) 1995 by MAXON-Computer
 * Autor: Markus Fritze
 * Compiler: Metrowerks CodeWarrior C++
 ***/

/***
 * globale Variablen
 ***/

// Versionsnummer vom System
short  gSystemVersion;

// true, wenn Programmende anliegt
Boolean  gQuitApplication;

// globaler Event-Record
EventRecord gTheEvent;

// Menü-IDs
enum {
 AppleMenuId = 128,
 FileMenuId,
 EditMenuId
};

/***
 * Betrag des Wertes ermitteln
 ***/
template<class T> T abs(T val)
{
 if(val < 0) return -val;
 return val;
}

/***
 * GetGestaltResult() ermittelt den Rückgabewert
 * der Gestalt Funktion. Falls ein Fehler auftritt,
 * so wird 0 zurückgegeben. Somit ist diese Funktion
 * nur dann brauchbar, wenn kein Fehler auftreten
 * kann (Gegenbeispiel: AUX-Version erfragen)
 ***/
static long GetGestaltResult(OSType gestaltSelector)
{
long gestaltResult;

 if (Gestalt(gestaltSelector,&gestaltResult) == noErr)
  return gestaltResult;
 else
  return 0;
}

/***
 * allgemeines Init
 ***/
static void InitAll(void)
{
 MaxApplZone();    // Zone auf volle Größe bringen
 for(int i = 3; --i>=0; )
  MoreMasters();   // Platz für ein paar Master-
  Pointer mehr

 InitGraf(&qd.thePort);  // QuickDraw init
 InitFonts();    // Font Manager init
 InitWindows();    // Window Manager init
 InitMenus();    // Menu Manager init
 TEInit();     // Text Edit init
 InitDialogs(nil);   // Dialog Manager Init
 InitCursor();    // Cursor auf Pfeil schalten

 // Systemversion erfragen
 gSystemVersion =
 GetGestaltResult(gestaltSystemVersion);

 qd.randSeed = TickCount(); // Zufallsbasis setzen

 // Menüzeile laden und aktivieren
 Handle hMbar = GetNewMBar(128);
 if(!hMbar) ExitToShell();
 SetMenuBar(hMbar);
 DisposeHandle(hMbar);

 // DAs ans Apfel Menü hängen
 AddResMenu(GetMHandle(AppleMenuId), 'DRVR');

 // Menüzeile zeichnen
 DrawMenuBar();
}

/***
 * Menüpunkte en- bzw. disablen
 ***/
static void AdjustMenu(void)
{
MenuHandle theMenu = GetMHandle(FileMenuId);

 // offenes Window vorhanden?
 if(FrontWindow())
  // dann "Schließen" enablen
  EnableItem(theMenu, 2);
```

→

```
102:  else
103:   // sonst: disablen
104:   DisableItem(theMenu, 2);
105: }
106:
107: /***
108:  * Menüpunkt wurde ausgewählt
109:  ***/
110: static void EventMenu(long m)
111: {
112: short title = HiWord(m);
113: short item = LoWord(m);
114:
115:  switch(title) {
116:  case AppleMenuId:
117:    if(item == 1) { // About…
118:     Alert(128, nil);
119:
120:    } else {
121:     // es wurde ein DA angewählt
122:     Str255  accName;
123:     GetItem(GetMHandle(AppleMenuId), item, accName);
124:     // DA öffnen
125:     OpenDeskAcc(accName);
126:    }
127:    break;
128:
129:  case FileMenuId: // Datei Menü
130:    switch(item) {
131:    case 1: // Neu
132:     // ein Fenster öffnen
133:     static Rect rBounds = { 40, 40, 140, 240 };
134:     NewWindow(nil, &rBounds, "\pTest", true,
135:       zoomDocProc, (WindowPtr)-1L, true, 0L);
136:     break;
137:    case 2: // Schließen
138:     DisposeWindow(FrontWindow());
139:     break;
140:    case 4: // Beenden
141:     gQuitApplication = true;
142:     break;
143:    }
144:    break;
145:
146:  case EditMenuId: // Bearbeiten Menü
147:    break;
148:  }
149:  // Invertierung des Menütitels zurücknehmen
150:  HiliteMenu(0);
151: }
152:
153: /***
154:  * Mausklick auswerten
155:  ***/
156: static void DoKlick(short dklick)
157: {
158:  WindowPtr theWindow;
159:  short  part = FindWindow(gTheEvent.where,
      &theWindow);
160:  switch(part) {
161:
162:  case inDesk:  // auf das Desktop o.ä. geklickt
163:  case inSysWindow:SystemClick(&gTheEvent, theWindow);
164:       break;
165:
166:  // Klick in die Menüleiste
167:  case inMenuBar: AdjustMenu();
168:       EventMenu(MenuSelect(gTheEvent.where));
169:       break;
170:
171:  // Klick in den Menütitel (draggen vom Window)
172:  case inDrag:  DragWindow(theWindow,
173:         gTheEvent.where, &qd.screenBits.bounds);
174:       break;
175:
176:  // Klick in den Sizer (Fenstergröße setzen)
```

```
177:  case inGrow:  // min. & max Größe vorgeben
178:       Rect sizeRect = { 40, 40, 300, 300 };
179:       long lSizeVH = GrowWindow(theWindow,
180:            gTheEvent.where, &sizeRect);
181:
182:       // Größe unverändert? => raus
183:       if(!lSizeVH) break;
184:
185:       // neue Fenstergröße setzen
186:       SizeWindow(theWindow,
187:         LoWord(lSizeVH), HiWord(lSizeVH), true);
188:
189:       // Window komplett neu zeichnen
190:       SetPort(theWindow);
191:       InvalRect(&theWindow->portRect);
192:       break;
193:
194:  // Klick in das Fenster
195:  case inContent: // Fenster getoppt?
196:       if(theWindow != FrontWindow()) {
197:        // nein => zuerst toppen!
198:        SelectWindow(theWindow);
199:       } else {
200:        // Klickanzahl * Beep...
201:        while(dklick-- > 0)
202:         SysBeep(10);
203:       }
204:       break;
205:
206:  // Klick in den Fuller
207:  case inZoomIn:
208:  case inZoomOut: // Maustaste im Fuller losgelassen?
209:       if(!TrackBox(theWindow, gTheEvent.where, part))
210:        break;
211:       // Window zoomen
212:       ZoomWindow(theWindow, part, true);
213:
214:       // Window komplett neu zeichnen
215:       SetPort(theWindow);
216:       InvalRect(&theWindow->portRect);
217:       break;
218:
219:  // Klick in den Closer
220:  case inGoAway: // Maustaste im Closer losgelassen?
221:       if(!TrackGoAway(theWindow, gTheEvent.where))
222:        break; // nein => raus
223:       // Fenster schließen
224:       DisposeWindow(theWindow);
225:       break;
226:  }
227: }
228:
229: /***
230:  * aktuellen Event verarbeiten
231:  ***/
232: static void DoEvent(void)
233: {
234: // letzter Maus-up-Event (Doppelklick-Erkennung)
235: static EventRecord gLastEvent;
236:
237: // in ein Register legen
238: register long m = gTheEvent.message;
239:
240:  switch(gTheEvent.what) {
241:  // Maustaste wurde losgelassen
242:  case mouseUp:
243:    // letzten Event für Doppelklick merken
244:    gLastEvent = gTheEvent;
245:    break;
246:
247:  // Maustaste wurde gedrückt
248:  case mouseDown: // mit Doppelklick-Verwaltung
249:    static short DClickCount;
250:
251:    // Test auf Mehrfachklick
```

→

```
252:     if(((gLastEvent.when + GetDblTime()) >
         gTheEvent.when) &&
253:       (abs(gTheEvent.where.h - gLastEvent.where.h) +
254:       abs(gTheEvent.where.v - gLastEvent.where.v)) <
           6) {
255:      DClickCount++;  // ein Mausklick mehr
256:     } else
257:      DClickCount = 1; // ein Mausklick
258:     DoKlick(DClickCount);
259:     break;
260:
261:   // Autorepeat auf Taste
262:   case autoKey: // wird normalerweise wie keyDown
       behandelt
263:     break;  // jedoch nicht bei der Menüauswahl
264:
265:   // Taste wurde gedrückt
266:   case keyDown:
267:     // Command gedrückt?
268:     if(!(gTheEvent.modifiers & cmdKey))
269:      break;   // nein => raus
270:     // Command + Taste auswerten
271:     AdjustMenu();
272:     EventMenu(MenuKey(m));
273:     break;
274:
275:   // Windowinhalt updaten
276:   case updateEvt:
277:     SetPort((WindowPtr)m);
278:     BeginUpdate((WindowPtr)m);
279:
280:     // Windowinhalt komplett löschen
281:     EraseRect(&((WindowPtr)m)->portRect);
282:
283:     // Sizer zeichnen
284:     DrawGrowIcon((WindowPtr)m);
285:
286:     // und ein graues Rechteck zeichnen
287:     static Rect theRect = { 20,20, 120, 120 };
288:     FillRect(&theRect, &qd.gray);
289:
290:     EndUpdate((WindowPtr)m);
291:     break;
292:
293:   // Window wurde getoppt oder untoppt
294:   case activateEvt:
295:     SetPort((WindowPtr)m);
296:     HiliteWindow((WindowPtr)m,
297:      (gTheEvent.modifiers & activeFlag) == activeFlag);
298:
299:     // Sizer zeichnen (oder löschen!)
300:     DrawGrowIcon((WindowPtr)m);
301:     break;
302:
303:   // Diskette wurde eingelegt
304:   case diskEvt:
305:     if(HiWord(m) == noErr) break; // kein Fehler?
306:
307:     // Vorschlag: Diskette formatieren
308:     static Point p = { 100, 100 };
309:     DIBadMount(p, m);
310:     break;
311:   }
312: }
313:
314: /***
315:   * main() hat jedes C Programm...
316:   *
317:   * Der Mac wertet jedoch nie einen Rückgabecode
318:   * aus, deswegen gibt sie nichts zurück.
319:   ***/
320: void  main(void)
321: {
322:   // Manager initialisieren
323:   InitAll();
324:
```

```
325:   // < System 7.0?
326:   if(gSystemVersion < 0x700)
327:     ExitToShell(); // => sofort beenden
328:   // besser: Fehlermeldung per Alert ausgeben...
329:
330:   // "Neu" im File Menü ausführen
331:   EventMenu(((long)FileMenuId « 16) | 1);
332:
333:   // Programmende abwarten...
334:   while(!gQuitApplication) {
335:     // Event abwarten
336:     if(WaitNextEvent(everyEvent, &gTheEvent, 5L, nil)) {
337:       // diesen abarbeiten
338:       DoEvent();
339:     }
340:   }
341: }
342:
343: /***
344:   * STC Demo.r
345:   *
346:   * Resource Datei in Textform.
347:   ***/
348:
349: #include "Types.r"
350: #include "SysTypes.r"
351:
352: resource 'MBAR' (128) {
353:   { /* array MenuArray: 3 elements */
354:     /* [1] */
355:     128,
356:     /* [2] */
357:     129,
358:     /* [3] */
359:     130
360:   }
361: };
362:
363: resource 'MENU' (128) {
364:   128,
365:   textMenuProc,
366:   0x7FFFFFFD,
367:   enabled,
368:   apple,
369:   { /* array: 2 elements */
370:     /* [1] */
371:     "Über STC Demo", noIcon, noKey, noMark, plain,
372:     /* [2] */
373:     "-", noIcon, noKey, noMark, plain
374:   }
375: };
376:
377: resource 'MENU' (129) {
378:   129,
379:   textMenuProc,
380:   0x7FFFFFFB,
381:   enabled,
382:   "Datei",
383:   { /* array: 4 elements */
384:     /* [1] */
385:     "Neu", noIcon, "N", noMark, plain,
386:     /* [2] */
387:     "Schließen", noIcon, "W", noMark, plain,
388:     /* [3] */
389:     "-", noIcon, noKey, noMark, plain,
390:     /* [4] */
391:     "Beenden", noIcon, "Q", noMark, plain
392:   }
393: };
394:
395: resource 'MENU' (130) {
396:   130,
397:   textMenuProc,
398:   0x7FFFFFFD,
399:   disabled,
```

→

```
400:  "Bearbeiten",
401:  { /* array: 6 elements */
402:   /* [1] */
403:   "Zurücknehmen", noIcon, "Z", noMark, plain,
404:   /* [2] */
405:   "-", noIcon, noKey, noMark, plain,
406:   /* [3] */
407:   "Ausschneiden", noIcon, "X", noMark, plain,
408:   /* [4] */
409:   "Kopieren", noIcon, "C", noMark, plain,
410:   /* [5] */
411:   "Einfügen", noIcon, "V", noMark, plain,
412:   /* [6] */
413:   "Alles auswählen", noIcon, "A", noMark, plain
414:  }
415: };
416:
417: resource 'ALRT' (128) {
418:  {40, 40, 176, 316},
419:  128,
420:  { /* array: 4 elements */
421:   /* [1] */
422:   OK, visible, silent,
423:   /* [2] */
424:   OK, visible, silent,
425:   /* [3] */
426:   OK, visible, silent,
```

```
427:   /* [4] */
428:   OK, visible, silent
429:  }
430: };
431:
432: resource 'DITL' (128) {
433:  { /* array DITLarray: 3 elements */
434:   /* [1] */
435:   {97, 197, 117, 255},
436:   Button {
437:    enabled,
438:    "OK"
439:   },
440:   /* [2] */
441:   {13, 13, 45, 45},
442:   Icon {
443:    disabled,
444:    1
445:   },
446:   /* [3] */
447:   {13, 59, 80, 255},
448:   StaticText {
449:    disabled,
450:    "STC Demo\n\n©1995 Ä-Soft, Markus Fritze"
451:   }
452:  }
453: };
```