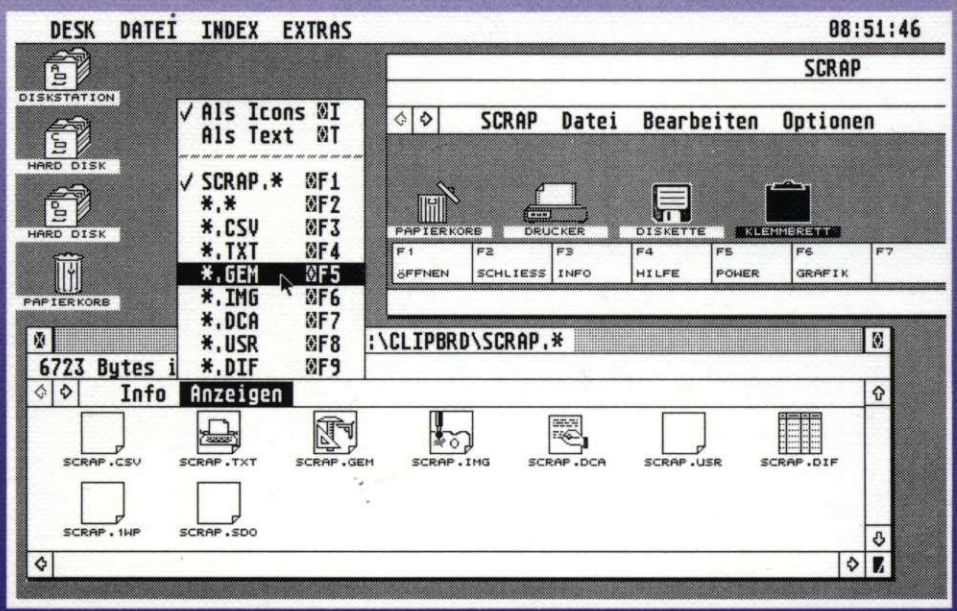
Dieter Geiß · Jürgen Geiß

# Vom Anfänger zum GEM-Profi

## Perfekte Programmierung auf Atari ST und IBM-PC

2., überarbeitete Auflage




Hüthig

Entwicklungstool auf Diskette

Dieter Geiß, Jürgen Geiß

# Vom Anfänger zum GEM-Profi

## Perfekte Programmierung auf Atari ST und IBM-PC

2., überarbeitete Auflage

Hüthig Buch Verlag Heidelberg

Diejenigen Bezeichnungen von im Buch genannten Erzeugnissen, die zugleich eingetragene Warenzeichen sind, wurden nicht besonders kenntlich gemacht. Es kann aus dem Fehlen der Markierung ® nicht geschlossen werden, daß die Bezeichnung ein freier Warenname ist. Ebensowenig ist zu entnehmen, ob Patente oder Gebrauchsmusterschutz vorliegen.

Alle Rechte an den Quellen zu den Beispielprogrammen in diesem Buch und auf der Begleitdiskette liegen bei der Hüthig Buch Verlag GmbH.

Käufer des Werkes sind berechtigt, alle aufgeführten Programme ohne Lizenzgebühren in eigene nicht gewerblich genutzte Anwenderprogramme zu integrieren, sofern die urheberrechtlichen Bestimmungen bezüglich der Zitierpflicht (z.B. der Vermerk „... enthält Programm xyz der Hüthig Buch Verlag GmbH" in sichtbarer Form) beachtet werden.

Jede kommerzielle Nutzung sowie der Verkauf der Quellen, auch in Teilen oder in übersetzter (compilierter) Form, ist untersagt. Zuwiderhandlungen werden strafrechtlich verfolgt und können daher zu schwerwiegenden wirtschaftlichen Konsequenzen führen.

Lizenzen für die kommerzielle Nutzung der Quellen können von der Hüthig Buch Verlag GmbH erworben werden. Stellen Sie dazu einen schriftlichen Antrag unter Angabe des Softwareprojekts. Für jedes Projekt muß eine eigene Genehmigung eingeholt werden.

Die Hüthig Buch Verlag GmbH übernimmt keine Garantien für die Programme und Informationen dieses Werkes, weder ausdrücklich noch unausgesprochen, in Bezug auf ihre Qualität, Durchführbarkeit oder Verwendbarkeit für einen speziellen Zweck. In keinem Fall ist die Hüthig Buch Verlag GmbH für direkte, indirekte, verursachte oder gefolgte Schäden haftbar, die aus der Anwendung der Software oder der Dokumentation resultieren.

CIP-Titelaufnahme der Deutschen Bibliothek

**Geiss, Dieter:**
Vom Anfänger zum GEM-Profi : perfekte Programmierung auf Atari ST und IBM-PC / Dieter Geiss ; Jürgen Geiss. – 2., überarb. Aufl. – Heidelberg : Hüthig, 1991
ISBN 3-7785-2049-0
NE: Geiss, Jürgen:

© 1991 Hüthig Buch Verlag GmbH, Heidelberg
Printed in Germany
Satz und Druck: Druckerei Bitsch GmbH, Birkenau

# Vorwort

Das Stichwort grafische Benutzeroberfläche ist aus dem Computeralltag kaum noch wegzudenken. Alles deutet darauf hin, daß eine neue Revolution der Mensch-Maschine-Schnittstelle auf die Softwareentwickler zukommt. Diese Tatsache kann man im Kampf ablesen, der vor allem auf dem Unix-Markt voll im Gange ist.

AT&T bringen mit Sun das Open-Look-System heraus, von OSF (Open Software Foundation) kommt das System Motif. Beide basieren auf dem System X-Windows. Microsoft schließlich kommt mit dem Presentation Manager (PM). Digital Research ist mit GEM schon seit einigen Jahren auf dem Markt, wobei GEM vor allem durch die Einführung des ATARI ST an Bekanntheit gewonnen hat.

Doch mit dem Vorhandensein einer grafischen Benutzeroberfläche allein ist noch lange nicht garantiert, daß ein Softwaresystem auch den Wünschen des Anwenders entspricht.

Die Entwicklung von Software unter grafischen Oberflächen setzt hauptsächlich 2 Komponenten voraus:

- das Programmer's Toolkit
- Richtlinien oder sogenannte Style Guidelines

Das Toolkit bildet den Kern der programmiertechnischen Seite. Mit dessen Hilfe können Softwarepakete effizient erstellt werden. Prototyping, Erstellung von Dialogboxen und Masken sowie die unabhängige Programmierung von internationalen Programmversionen werden durch das Toolkit erst ermöglicht.

Größte Bedeutung hat aber neben den programmiertechnischen Voraussetzungen das Vorhandensein von Style Guidelines. Sie garantieren, daß Softwarepakete nicht das Produkt aus spontaner Kreativität und Willkür einzelner Programmierer sind. Sie sorgen für Konformität auch unter einer größeren Palette von Applikationen verschiedener Hersteller.

Style Guidelines minimieren auch die Entwicklungszeiten für Oberflächen, da sich Programmierer nicht mehr mit langwierigen Überlegungen herumschlagen müssen, wie Pull-Down-Menüs, Dialogboxen und Interaktionen mit Bildschirmfenstern auszusehen haben.

Durch das Benutzen von Guidelines bei der Softwareentwicklung wird gewährleistet, daß der Anwender über verschiedene Applikationen hinweg auf bereits gewonnene Erfahrungen zurückgreifen kann, was wiederum zu einer hohen Akzeptanz der Software führt.

Nur Apple ist es bisher gelungen, Richtlinien für Entwickler bereitzustellen, um die Konsistenz zwischen verschiedenen Applikationen sicherzustellen. Diese Richtlinien sind in dem Werk INSIDE MACINTOSH festgehalten. Wer als Anwender einmal auf einem Macintosh gearbeitet hat, wird mit dem Erlernen von neuen Applikationen kaum noch Probleme haben.

Ein entsprechendes Werk für GEM, das von Digital Research oder ATARI herausgegeben wurde, war bisher noch nicht in Sicht. Es ist Aufgabe dieses Buches, das Fehlende jetzt nachzuholen. Deshalb können Sie dieses Werk als INSIDE GEM ansehen, das Softwareentwicklung unter GEM inklusive Theorie der Benutzeroberflächen vollständig beschreibt. Es entstand in Zusammenarbeit mit ATARI Computer GmbH und Digital Research. Das neue Programmer's Toolkit von DRI enthält auch die Programmierumgebung für die verschiedenen Systeme (PORTAB.H, VDI.H und AES.H), während ATARI die Regeln für die Gestaltung von GEM-Software voll übernommen hat und dieses Werk jedem Softwareentwickler ans Herz legt.

Das Buch enthält eine komplette Programmierumgebung für GEM-Software, so daß Entwickler nur noch einen minimalen Aufwand betreiben müssen, um perfekte Software unter GEM zu erstellen. Sie müssen lediglich die Funktionalität Ihrer Applikation an das vorhandene Skelett-Programm SCRAP anhängen und die Regeln in Kapitel 5 beachten. Die komplexe Steuerung von Software unter einer grafischen Oberfläche wird Ihnen dabei komplett abgenommen.

# Inhalt


| | | |
|---|---|---|
| **1** | **Einleitung** | 11 |
| **2** | **GEM und sein Umfeld** | 17 |
| 2.1 | Versionen | 18 |
| 2.1.1 | GEM 1.X | 18 |
| 2.1.2 | GEM 2.X | 20 |
| 2.1.3 | GEM/3 bzw. GEM 3.X | 21 |
| 2.1.4 | X/GEM | 21 |
| 2.1.5 | Zukünftige Versionen | 22 |
| 2.2 | Struktur | 22 |
| 2.3 | VDI | 25 |
| 2.3.1 | Koordinatensysteme | 25 |
| 2.3.2 | Aufbau des VDI | 27 |
| 2.3.3 | VDI-Funktionsgruppen | 28 |
| 2.3.4 | ASSIGN.SYS | 31 |
| 2.3.5 | Kommunikation mit dem VDI | 39 |
| 2.3.6 | Ein erstes VDI-Beispiel | 42 |
| 2.3.7 | Funktionen der VDI-Gruppen | 56 |
| 2.4 | AES | 96 |
| 2.4.1 | Speicherbelegung | 96 |
| 2.4.2 | AES-Komponenten | 97 |
| 2.4.3 | AES-Funktionsgruppen | 98 |
| 2.4.4 | Kommunikation mit dem AES | 103 |
| 2.4.5 | Allgemeine Aufruffolgen im AES | 107 |
| 2.4.6 | Vorgehensweise beim Aufbau eines GEM-Programms | 115 |
| 2.4.7 | Funktionen der AES-Gruppen | 116 |
| 2.5 | Resourcen | 162 |
| 2.5.1 | GRC | 163 |
| 2.5.2 | RCM | 175 |
| 2.6 | Accessories | 183 |
| 2.7 | Formate | 183 |
| 2.7.1 | Metadatei-Format | 184 |

| | | |
|---|---|---|
| 2.7.2 | Bit-Image-Format | 198 |
| 2.7.3 | OUT-Dateiformat | 207 |
| **3** | **Entwicklungsumgebungen** | 209 |
| 3.1 | ATARI GEMDOS | 221 |
| 3.1.1 | Digital Research C | 222 |
| 3.1.2 | Laser C | 224 |
| 3.1.3 | Lattice C | 226 |
| 3.1.4 | Mark Williams C | 228 |
| 3.1.5 | Turbo C | 229 |
| 3.2 | MS-DOS und FlexOS | 232 |
| 3.2.1 | Microsoft C | 232 |
| 3.2.2 | Turbo C | 235 |
| 3.2.3 | High C | 236 |
| **4** | **Aufgabenstellung eines kleinen GEM-Programmes** | 239 |
| 4.1 | Die Bedienung von SHOWGEM | 240 |
| 4.2 | Erläuterungen zum Programm SHOWGEM | 242 |
| **5** | **Gestaltung von Benutzeroberflächen** | 289 |
| 5.1 | Theorie | 290 |
| 5.1.1 | Die Mensch-Maschine-Schnittstelle | 290 |
| 5.1.2 | Das Modell „Mensch-als-Prozessor“ | 291 |
| 5.1.3 | Weitere Gesetze und Prinzipien | 297 |
| 5.2 | Folgerungen | 302 |
| 5.2.1 | Menüs | 303 |
| 5.2.2 | Mausbewegungen | 310 |
| 5.2.3 | Übung | 313 |
| 5.2.4 | Aufmerksamkeit | 314 |
| 5.2.5 | Vorhersehbarkeit und Modi | 314 |
| 5.2.6 | Übersichtlichkeit von Dialogboxen | 322 |
| 5.3 | Fehlerbehandlung | 325 |
| 5.4 | Hilfe | 326 |
| 5.5 | Datenaustausch | 329 |
| 5.6 | Ausgabe von Daten | 336 |
| 5.7 | Allgemeines | 337 |
| **6** | **Ein universelles Modulkonzept in C** | 339 |
| 6.1 | Ziel der Beispielapplikation SCRAP | 339 |
| 6.2 | Aufbau und Aufgaben der Module | 341 |
| 6.3 | Allgemeine Fensterverwaltung | 355 |
| 6.4 | Programmierumgebungen für SCRAP | 378 |
| 6.4.1 | GEMDOS | 379 |
| 6.4.2 | MS-DOS | 384 |
| 6.4.3 | FlexOS | 388 |
| 6.5 | Modulbeschreibung | 390 |

6.5.1 Modul GEMAIN ..... 397
6.5.2 Modul GLOBAL ..... 400
6.5.3 Modul WINDOWS ..... 428
6.5.4 Modul RCM ..... 451
6.5.5 Modul EVENT ..... 451
6.5.6 Modul INITERM ..... 454
6.5.7 Modul RESOURCE ..... 456
6.5.8 Modul MENU ..... 458
6.5.9 Modul DESKTOP ..... 460
6.5.10 Modul CLIPBRD ..... 464
6.5.11 Modul DISK ..... 468
6.5.12 Modul PRINTER ..... 469
6.5.13 Modul TRASH ..... 470
6.5.14 Modul IMAGE ..... 472
6.5.15 Modul META ..... 473
6.5.16 Modul EDIT ..... 475
6.5.17 Modul POWER ..... 476
6.5.18 Modul GRAF ..... 478
6.5.19 Modul MODULE ..... 480
6.6 Bedienung von SCRAP ..... 488

**Anhang**
A. Header-Dateien ..... 491
B. Make-Dateien ..... 523
C. Die neue Software ..... 539

**Stichwortverzeichnis** ..... 573

# Danksagung

An dieser Stelle möchten wir uns bei all denjenigen bedanken, die zur Erstellung des Buches beigetragen haben oder die die Verwirklichung der Idee einer einheitlichen Oberfläche unterstützt haben. Auch der kleinste Beitrag half uns oft ein großes Stück weiter. Die Namen sind alphabetisch geordnet, weil jede(r) seinen/ihren Beitrag geleistet hat. Wir danken:

Arnd Beißner, für die zur Verfügungstellung des AMC-GDOS auf dem ATARI ST, ohne das die Entwicklung von SCRAP und SHOWGEM nicht möglich gewesen wäre.

Unseren Frauen Christina und Monika Geiß, die nicht nur viel Zeit aufgebracht haben, um das gesamte Werk zu korrigieren und Zeichnungen zu erstellen, sondern weil sie in der über 1-jährigen Entwicklung an vielen Abenden und Wochenenden auf uns verzichten mußten.

Oliver Joppich, Application Systems Heidelberg, der uns immer schnellstmöglich die neueste Version von Laser C zur Verfügung gestellt hat, mit dem wir die Demoprogramme testen konnten.

Thomas Klingelhöfer für die zur Verfügungstellung eines Algorithmus zum Lesen und Darstellen von monochromen Bit-Image-Dateien, der uns als Beispiel für die Farbversion diente.

Harald Müller, ATARI Computer GmbH, der uns half, bei ATARI ein offenes Ohr zu finden. Er fand solche Übereinstimmung mit seinen eigenen Ideen, daß er unsere Vorschläge als Style Guides voll übernahm.

Julian F. Reschke, Mitarbeiter der Redaktion ST-Magazin, der in fast jeder Ausgabe Softwareentwickler zum Umdenken anregt, endlich Standards zu benutzen. Auch er stimmt mit uns überein, wenn es um das Setzen von Standards geht.

Robert Schneider, Digital Research GmbH in München, der sich dafür eingesetzt hat, daß unsere Entwicklungsumgebung in den neuesten Versionen des Programmer's Toolkit Verwendung findet und der für GEM-Probleme immer ein offenes Ohr hat. Er unterstützte uns bei der Implementierung unserer Routinen auf MS-DOS.

Ingo Weber, Siemens AG, der uns wertvolle Tips gab, unsere Ideen auch unter X/GEM zu testen.

Peter Weinholz, SciLab GmbH, der uns nicht nur bei Problemen mit Metadateien half, sondern die Idee der Standardisierung voll mit unterstützt. Er brachte uns auf einige gute Ideen, wie Fenster mit eigenen Mausformen oder der Klemmbrettverwaltung.

# 1 Einleitung

Seit dem Erscheinen des Atari ST auf dem deutschen Markt wurden mehrere hunderttausend Geräte abgesetzt. Diese Beliebtheit ist einerseits dem günstigen Preis-Leistungs-Verhältnis zuzurechnen, andererseits aber auch der einfachen Bedienung. Sie resultiert aus dem Vorhandensein der grafischen Benutzeroberfläche GEM (Graphics Environment Manager), die beim Atari ST standardmäßig mitgeliefert wird.

So können auch Personen, die noch nie oder sehr selten mit Rechnern umgegangen sind, die Berührungsängste genommen werden. Beispielsweise werden Atari-Rechner wegen der leichten Bedienbarkeit bestimmter Programme sogar beim deutschen Bundesgerichtshof eingesetzt.

Das GEM der Firma Digital Research Incorporated, welche manchmal auch DRI genannt wird, war bereits vorhanden, bevor es den Atari ST gab. Es wurde ursprünglich für IBM PC und Kompatible geschaffen. Atari hat dann die Lizenzen für eine Version für ihren ST eingekauft. Auf dem IBM PC fristete GEM jedoch ein eher stiefmütterliches Dasein, da es erstens extra gekauft werden mußte und es zweitens nicht genügend gute Software gab.

Es ist nun zu beobachten, daß immer mehr Programme, die ursprünglich für den Atari entwickelt werden, auch auf dem IBM laufen können. Allerdings wurde das GEM für die PCs weiterentwickelt, so daß es nun eine Reihe von verschiedenen Versionen gibt. Genannt seien hier GEM 2.1, GEM 2.2 oder GEM/3. Letzteres liegt inzwischen wieder in der Version GEM 3.11 vor. Ganz neu ist X/GEM. Dieses Extended GEM ist eine grafische Oberfläche, die nur unter einem Multitasking-Betriebssystem läuft, also z.B. FLEXOS (von DRI). Dieses bietet dann aber auch Möglichkeiten, von denen Atari-Besitzer nur träumen können. Mehrere Prozesse können in eigenen Fenstern parallel ablaufen (oder wenigstens quasiparallel bei Einsatz nur eines Mikroprozessors).

Demgegenüber hat Atari keine neue GEM-Version von DRI gekauft, so daß auf diesem Rechner immer noch das „alte" GEM läuft (Versionsnummer kleiner 2).

Während der Atari ST von Haus aus drei verschiedene Auflösungen bietet, muß beim GEM für den PC beim Installieren angegeben werden, welche Grafikkarte man sein eigen nennt. Außerdem gibt es noch die Möglichkeit, verschiedene Großbildschirme an den Rechner anzuschließen. Es ist wohl offensichtlich, daß GEM-Software zumindest so entwickelt werden muß, daß sie möglichst mit jeder Bildschirmauflösung läuft.

Für den neuen Atari TT gibt es sogar noch mehr Bildschirmauflösungen. Um Programme auf diesen Rechner anzupassen, müssen sie schon einigermaßen sauber programmiert sein.

Desweiteren kann man noch unterscheiden, ob ein Programm als Haupt-Applikation im Vordergrund läuft oder als Accessory im Hintergrund. Accessories haben normalerweise keine eigene Menüzeile und keinen eigenen Desktop. Beim X/GEM entfällt dies, da jedes Programm im Vorder- oder Hintergund ablaufen kann.

Ein weiterer Stolperstein kann der Einsatz der Entwicklungswerkzeuge sein. Geht man beispielsweise von der Sprache „C" aus, in der auch das GEM entwickelt wurde, so findet man eine Vielzahl von Compilern auf dem jeweiligen Rechner. Während sich beim IBM alle Compiler einigermaßen an den Microsoft-Standard halten, kocht beim Atari jeder sein eigenes Süppchen. Sogar einige Systemaufrufe der GEM-Bibliotheken haben verschiedene Namen, von den Inkompatibilitäten der erzeugten Objekt-Dateien einmal ganz zu schweigen.

Es scheint nun geradezu unmöglich, GEM-Software so zu entwickeln, daß sie auf jedem Rechner mit jedem Betriebssystem, mit jeder Auflösung, mit jeder GEM-Version, als Programm oder Accessory mit jedem Compiler läuft.

Wünschenswert wäre eine Methode, Software so zu entwickeln, daß sie von selbst erkennt, in welcher Umgebung und wie sie abzulaufen hat, wobei die Software nicht modifiziert werden muß. Beim Compilieren sollte außerdem der Quelltext nicht geändert werden müssen, wenn von einem Compiler auf einen anderen oder von einem Betriebssystem auf ein anderes gewechselt wird.

Sie sind sicher interessiert daran, Ihre Software genauso zu entwickeln und suchen dafür eine Methode. Die in diesem Buch dargestellte Methode erfüllt genau die oben angegebenen Forderungen. Wenn Ihnen das unglaubhaft erscheint, so können wir Ihnen nur empfehlen, das Buch durchzuarbeiten. Dabei spielt es keine Rolle, ob Sie Anfänger oder Fortgeschrittener sind, da wir von vorne anfangen werden. Die einzigen Voraussetzungen sind:

- Kenntnis der Programmiersprache C
- Ein C-Compiler
- Das GEM-Entwicklungspaket (nur für IBM PC)

Die Sprache „C" ist Voraussetzung, da alle Quellen in „C" verfaßt wurden. Das Lernen der Sprache können wir Ihnen leider nicht beibringen. Aber es gibt bereits unzählige Bücher zu diesem Thema, z.B.:

Mit C zum Ziel
Arnold Hickersberger
Hüthig Verlag
ISBN 3-7785-1555-1

Ein Nachschlagewerk ist notwendig, da nicht noch einmal alle Funktionen (ungefähr 200) erklärt werden sollen. Auch Nachschlagewerke gibt es viele. Ein Vorschlag könnte sein:

Softwareentwicklung auf dem Atari ST, 2.Aufl.
Jürgen Geiß/Dieter Geiß
Hüthig Verlag
ISBN 3-7785-1533-0

Im Prinzip können Sie den C-Compiler benutzen, der Ihnen am besten gefällt. Allerdings wurde die Lauffähigkeit des vorgestellten Programms nur für folgende Compiler getestet:

Auf ATARI ST unter GEMDOS:

- Digital Research C (Alcyon C)
- Mark Williams C
- Megamax Laser C
- Lattice C
- Turbo C

Auf IBM PC unter MS-DOS:

- Microsoft C
- Turbo C

Auf 386er Rechnern unter FlexOS:

- Metaware High C

Leider mußten wir feststellen, daß einige Compiler nicht immer korrekten Code erzeugten oder wegen anderer Mängel unbrauchbar sind. Genaueres finden Sie in Kapitel 3.

Wenn Sie Ihren Lieblings-Compiler gefunden haben, können Sie beruhigt weiterlesen. Wenn nicht, sollten Sie überlegen, ob Sie auf einen anderen Compiler umsteigen sollten. Es ist meistens sehr schwierig, Fehler des Compilers zu finden.

Entwicklen Sie unter MS-DOS, so müssen Sie noch das Entwicklungspaket von Digital Research erwerben. Beim Atari sind bei allen Compilern die GEM-Aufrufe bereits in den Bibliotheken enthalten. Ein Resource-Construction-Set sollte ebenfalls erworben werden.

Es ist bei den meisten Compilern aber ebenfalls Bestandteil der Entwicklungsumgebung.

Welches Projekt erwartet Sie nun?

Als Beispielprogramm haben wir eine Applikation vorgesehen, die ein Klemmbrett auf Diskette verwaltet. Dieses Klemmbrett soll in der Lage sein, die wichtigsten Standardformate direkt auszugeben. Dazu gehören normale ASCII-Dateien, GEM Metafiles und Bit-Image-Files. Zusätzlich sind noch einige Demo-Optionen in das Programm eingebaut.

Es erwarten Sie außer den schon oben genannten noch weitere interessante Themen, die sogar Profis aufhorchen lassen müßten. In den einzelnen Kapiteln wird auf Folgendes eingegangen.

Kapitel 2 beschäftigt sich mit GEM. Es ist vor allem für Anfänger gedacht, die noch keinen tieferen Einblick in die GEM-Struktur haben. Es behandelt aber auch die Implementierung der verschiedenen Versionen und beschäftigt sich mit dem Aufbau der Resourcen (Dialogboxen, Menüs etc.). Falls Sie Ihre Resourcen nur auf einem Rechner erstellen und auf einen anderen Rechner überspielen wollen, müssen Sie diese erst konvertieren. Dafür wird aber ein Programm vorgestellt werden, das diese Konvertierung automatisch erledigt. Es nennt sich GRC (GEM Resource Converter).

In Kapitel 3 wird auf die Entwicklungsumgebung auf verschiedenen Rechnern mit den verschiedenen Compilern eingegangen. Dort werden auch Begriffe wie Portabilität und ANSI-C erklärt. Sie werden erkennen, warum es besser ist, mit einem ANSI-Compiler zu arbeiten. Es wird eine Methode vorgestellt, wie man Header-Dateien schreibt, die von ANSI und Nicht-ANSI-Compilern verstanden werden. Außerdem soll in diesem Kapitel der MAKE-Mechanismus erklärt werden. Ursprünglich aus UNIX kommend greift dieser Mechanismus nun auch auf PCs immer weiter um sich. Anfängern bietet er oft Schwierigkeiten. MAKE unterstützt die Software-Erstellung größerer Projekte ungemein.

Erste Programmierschritte werden in Kapitel 4 getan. Dieses ist vor allem für Anfänger gedacht.

In Kapitel 5 wird es zunächst theoretisch. Es geht um die Gestaltung von Benutzeroberflächen. Warum ist ein Programm leichter zu bedienen als ein anderes? Gibt es Regeln für die Gestaltung von Oberflächen, die garantieren, daß ein Programm übersichtlich und leicht bedienbar wird? Aus benutzerpsychologischer Sicht wird der Mensch vor dem Rechner betrachtet. Welche Aktionen laufen ab, wenn ein Mensch Menüs betrachtet oder in eine Dialogbox klickt? Aus der Theorie lassen sich zahlreiche Folgerungen für die Praxis ableiten. Dieses Kapitel wendet sich auch an den fortgeschrittenen Leser und setzt keine Programmierkenntnisse voraus. Es ist daher auch für diejenigen gedacht, die nicht mit „C", sondern einer anderen Sprache arbeiten.

Ein universelles Modulkonzept soll in Kapitel 6 erläutert werden. Es zeigt, wie man Module in „C" schreibt. Dabei hat jedes Modul eine spezifische Aufgabe. Der große Überhang, der in GEM-Programmen entsteht, soll vermieden werden. Warum das Rad immer

neu erfinden? Hat man eine Menge von Modulen, so kann man sich dann aus diesen sein Programm zusammensetzen. Verfechter von Modula müssen spätestens nach Lesen dieses Kapitels zugeben, daß modulare Programmierung auch in „C" sehr sauber möglich ist.

Weiterhin werden alle Module des Demoprogramms erklärt, aber nur die wichtigsten aufgelistet. Es wurden Routinen implemenentiert, die z.B. Pop-Up-Menüs oder Menüzeilen in Fenstern möglich machen. Auch Accessories dürfen einen eigenen Desktop haben. Dieser wird dann in ein Fenster gelegt. Die entsprechenden Routinen werden jeweils genau erläutert.

Im Anhang werden Header-Dateien aufgelistet, die für jeden Compiler und jedes Betriebssystem gleich sind! Dabei wird Portabilität großgeschrieben. Dazu kommen Listings der Make- bzw. Projekt-Dateien.

Noch ein Wort zur Einführung: Würde jeder Software-Entwickler nach der in diesem Buch beschriebenen Methode arbeiten, könnte der Käufer der Software jederzeit entscheiden, in welcher Auflösung er ein Programm bedienen möchte und ob er das jeweilige Programm als Accessory im Hintergrund oder als Hauptprozeß im Vordergrund laufen lassen möchte (oder beides). An der Bedienung würde sich praktisch nichts ändern. Damit wäre ein Art Multitasking sogar auf dem Atari möglich! Schließlich könnte er sich entscheiden, ob er das Programm lieber zu Hause auf einem Atari ST oder TT einsetzen möchte oder in der Firma auf einem IBM PC.

# 2 GEM und sein Umfeld

GEM ist die Abkürzung für „Graphics Environment Manager“ Für alle diejenigen die es falsch aussprechen: es kommt aus dem englischen und bedeutet auch Juwel, woher sich auch das GEM-Logo ableitet. Bei GEM handelt sich um ein System, das die Erstellung und den Ablauf von Software mit einer modernen Benutzerschnittstelle ermöglicht. Solche Software zeichnet sich dadurch aus, daß die Interaktion mit dem Endbenutzer über eine sogenannte grafische Benutzeroberfläche stattfindet. Merkmale hiervon sind Mausbedienung, Pull-Down/Pop-Up-Menüs, Bildschirm-Fenster und Piktogramme (Icons).

In den kommenden Jahren wird die Bedienung von Softwaresystemen zunehmend von diesen Oberflächen geprägt werden. Der Grund dafür liegt in der schnelleren Begreifbarkeit und der einheitlichen Bedienung auch verschiedener Programme. Der Benutzer kann sich so auf das Lösen seiner Aufgabe konzentrieren und nicht darauf, wie er sie zu bewältigen hat.

Auf den verschiedenen Rechnertypen haben sich einige dieser Systeme herauskristallisiert. So finden wir auf dem Apple Macintosh das erste kommerzielle System einer grafischen Benutzeroberfläche. Da Apple selbst Forschung in dieser Richtung betreibt, kann dieses System als das ausgereifteste gelten.

Auf UNIX-Systemen wird sich wahrscheinlich ein System namens X-Windows durchsetzten. Es bietet ebenfalls eine grafische Benutzeroberfläche und sogar die Möglichkeit mehrere Prozesse in verschiedenen Bildschirmfenstern auf dem gleichen Terminal (Server) laufen zu lassen. Programme (Clients) können auf der anderen Seite der Welt gestartet werden und die Kontrolle mittels Nachrichten über das Grafikterminal übernehmen. Man nennt dies dementsprechend Client-Server-Modell. Der Server übernimmt hier den gesamten geräteabhängigen Bildschirmaufbau. Interessant ist in diesem Zusammenhang, daß ATARI mit seiner TT-Rechnerserie UNIX mit X-Windows ausliefern wird. Auch die Transputer-Workstation von Atari (ATW) läuft unter dem Betriebssystem Helios mit X-Windows.

Auf IBM-PCs und Kompatiblen gibt es schon seit einigen Jahren GEM, welches von Digital Research entwickelt wurde. Doch dazu mehr in Kapitel 2.1. Ein anderes System stammt von Microsoft, dem Marktführer der Software-Entwicklungsschmiede. Es han-

delt sich um MS-Windows, das aber erst in den letzten Monaten mehr an Bedeutung gewinnen konnte. Die bisher schwache Verbreitung liegt zum einen an Anfangsschwierigkeiten bei der Auslieferung der ersten Versionen und damit einhergehend dem schwachen Software-Angebot, das MS-Windows voll ausschöpft. Wahrscheinlich wird sich dieses System auf IBM-PCs und Kompatiblen durchsetzen, da Microsoft „die besten Karten" auf diesen Rechnern hat.

Anders hingegen sieht es bei nichtkompatiblen PCs aus. Dort gibt es im wesentlichen nur noch 3 Typen. Der o.g. Apple Macintosh, der Commodore Amiga und der Atari ST. Der Amiga bedient sich ebenfalls einer grafischen Bernutzeroberfläche, die Intuition genannt wird. Wer sich dafür weiter interessiert, sollte in entsprechender Literatur nachschlagen.

Übrig bleibt noch der ATARI ST. Die Chefentwickler bei ATARI entschieden sich für GEM, das die einzige grafische Benutzeroberfläche war, die bereits lauffähig existierte und nur noch vom PC auf den ATARI portiert werden mußte. Das Betriebssystem wurde selbst entwickelt und bekam den Namen TOS (The Operating System), das sich aus dem BIOS, dem XBIOS, dem GEMDOS sowie dem GEM und dem Desktop zusammensetzt. Diese 5 Teile wurden dann auch in das 192 KByte fassende ROM gebrannt.

Wenn vorhin von einheitlicher Bedienung auch verschiedener Programme gesprochen wurde, so trifft dies nur auf den Apple Macintosh zu. Die Firma Apple ist die einzige, die Richtlinien erlassen hat, an die sich Software-Entwickler zu halten haben. So sehen auch Macintosh-Programme immer ähnlich aus und sind gleich zu bedienen. Solche Software kann auch unter GEM entwickelt werden. In den folgenden Kapiteln dieses Buches wird gezeigt, wie man das macht.

## 2.1 Versionen

Zur Zeit exisitieren 4 unterschiedliche GEM-Versionen (ohne Nebenversionen). Zu erwähnen ist vielleicht noch der Vorgänger von GEM. Es handelt sich um das GSX-System (Graphics Extension), welches Grafikroutinen zur Verfügung stellt, auf die man als Programmierer zurückgreifen kann und die die Erstellung von Grafik-Applikationen erleichtern sollten.

### 2.1.1 GEM 1.X

Diese Version ist auf PC's kaum noch anzutreffen, da sie schon sehr alt ist und durch neue Versionen ersetzt wurde. Dafür hat sie aber eine andere entscheidende Bedeutung. Es ist die Version, die im TOS des ATARI ST seinen Dienst erledigt und das ungefähr bei 300000 ATARI ST Kunden. Diese Version wurde seit dem ersten ROM-Release kaum weiterentwickelt. Lediglich einige Bugs wurden behoben. In der neuesten TOS Version 1.4 wurden auch einige wenige Funktionen hinzugefügt. Sie werden im Teil VDI und AES beschrieben.

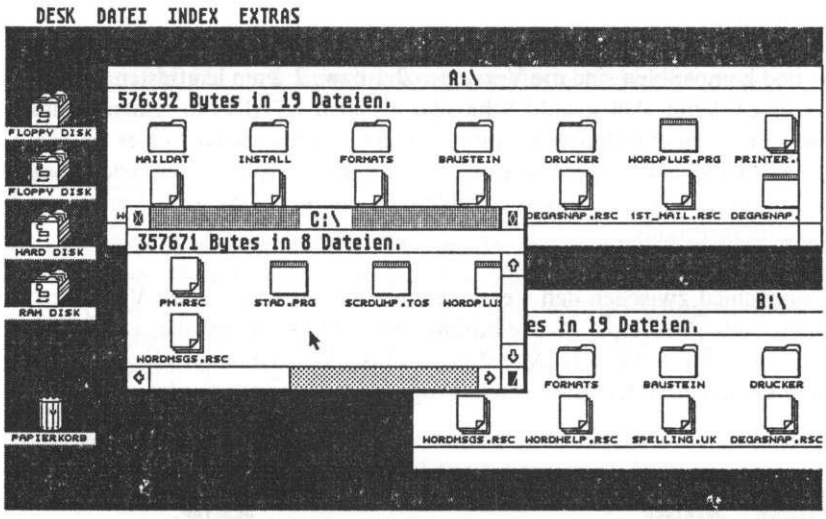



**Abb. 2.1:** GEM 1.X Desktop (ATARI ST)

Ein Vorteil der alten Version ist die Portierbarkeit. Wird GEM-Software auf einem ATARI ST entwickelt, so läuft sie auch unter neueren GEM-Versionen, wenn man sich an die Schnittstellen hält. Lediglich die GEM-Erweiterungen des TOS 1.4 sind nicht mit GEM 2.X-Versionen kompatibel. Das liegt daran, daß das TOS 1.4 von ATARI weiterentwickelt und gepflegt wird, während die Pflege aller anderen GEM-Versionen vom Erfinder Digital Research vorgenommen wird.

Ein großes Manko hat allerdings diese GEM Version. Da der Platz im ROM nicht ausgereicht hat, wurde ein wichtiger Teil des GEM's ausgelagert. Es handelt sich um das GDOS (Graphics Device Operating System). Es beinhaltet die geräteunabhängigen Grafikfunktionen. Sie wurden auf dem ATARI ST nur für den Bildschirm implementiert und müssen für Drucker, Kamera, Plotter und Metadateien nachgeladen werden. Dabei spielt die Datei ASSIGN.SYS eine entscheidende Rolle. Sie wird in Kapitel 2.3 erklärt.

Das Fehlen des GDOS für andere Ausgabegeräte als den Bildschirm (z.B. Drucker, Metadateien), sowie des Programmes OUTPUT, welches grafische Informationen zu Papier bringt, hat Schuld daran, daß ATARI Softwareprodukte untereinander kaum Informationen austauschen können. So existieren für fast jedes Textprogramm eigene Druckertreiber und für fast jedes Grafikprogramm eigene Grafikformate. ATARI hat hier den entscheidenden Fehler gemacht und bei der Auslieferung der Entwicklungspakete die Standardgrafikformate von GEM (Meta- und Bit-Image-Dateien) durch das Fehlen des GDOS und des OUTPUT-Programmes nicht unterstützt.

### 2.1.2 GEM 2.X

Auf IBM-PCs und kompatiblen sind die Versionen 2.1 bzw. 2.2 am häufigsten vertreten. Dies liegt zum einen daran, daß sowohl Schneider als auch ATARI mit seinen kompatiblen PCs diese Versionen zur Hardware mit ausliefern. Zum anderen gibt es Software-Produkte wie ADIMENS GT, Superbase usw., welche ebenfalls diese Versionen kostenlos dem Produkt beifügen. Lizenzen kosten außerdem nur 1.– DM für die ersten 1000 Stück pro verkauftem Produkt.

Der größte Unterschied zwischen den Versionen 1.X und allen weiteren Versionen ist die Aufmachung des Desktops (Schreibtisch). Der Desktop ist die erste GEM-Applikation, die beim Starten von GEM auf dem Bildschirm erscheint. Mit ihm können Dateien einfach kopiert und gelöscht, sowie Programme gestartet werden.

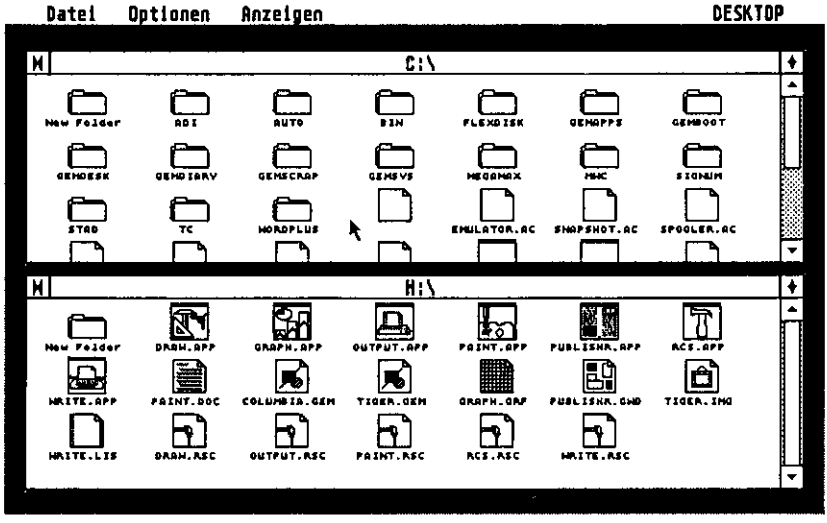



**Abb. 2.2:** GEM 2.X Desktop (ATARI ST/IBM PC)

Da Digital Research einen Prozeß gegen Apple Computer verlor, mußten sie das Erscheinungsbild des Desktop-Programms ändern. Standen in der Version 1.X noch vier frei bewegliche Fenster für Dateimanipulationen zur Verfügung, so waren es ab Version 2.X nur noch 2 feste Fenster. Dies veranlaßte die ATARI-Softwareentwickler leider zu der Meinung, daß das gesamte GEM-System nur noch 2 feste Fenster verwalten könne; daß der Desktop nur eine Applikation ist, die eben nur 2 feste Fenster öffnet, mit denen man arbeiten kann, bemerkten nur die wenigsten.

Vielleicht ist dies auch ein Grund, warum sich die GEM Version 2.2 auf dem ATARI ST nie durchsetzen konnte. Angeboten wird sie jedenfalls von der Firma ABC Software zum Preis von ca. 169.– DM, die auch die Anpassung der Version vom PC auf den

ATARI vornahm. Einen Vorteil allerdings besitzt diese ATARI-Version. Das GDOS ist fester Bestandteil und muß nicht zusätzlich erst noch geladen werden. So kann Software von einem IBM PC ohne Einschränkung auf einen ATARI ST portiert werden.

### 2.1.3 GEM/3 bzw. GEM 3.X

Sei ca. März '87 existiert die GEM/3-Version. Sie wurde so erweitert, daß der Ventura-Publisher (ab GEM 3.11), ein Desktop-Publishing-System von XEROX, nun ohne spezielles GEM-System auskommt. Beim Ventura-Publisher wurde nämlich eine spezielle GEM-Version mitgeliefert, da die Funktionalität der GEM-2.X-Versionen den Fähigkeiten des Publishers nicht Rechnung tragen konnte.

Sonst hat sich nicht viel geändert. Zeichensätze können jetzt dynamisch in Overlays geladen werden, was für den knappen Speicher bei einer MS-DOS-Maschine mit 640 KByte wichtig ist. ATARI ST Besitzer können da nur lächeln, da Sie bei einem MEGA ST 4 mit 4 Megabyte Speicher reichlich Platz für GEM-Programme haben.

Auch die Installierung (GEMPREP.BAT) wurde wesentlich verbessert. Man kann z.B. jetzt mehrere verschiedene Druckertypen gleichzeitig benutzen (OUTPUT zeigt diese mit Namen an). So könnte man Programmlistings auf einem preiswerten Nadeldrucker ausgeben, während man für Publikationen einen Laserdrucker benutzt.

### 2.1.4 X/GEM

X/GEM ist das, wovon jeder ATARI ST Besitzer schon immer geträumt hat. Es handelt sich um eine GEM-Version, die auf einem Multiuser-Multitasking Betriebssystem implementiert wurde. X/GEM erlaubt die gleichzeitige Abarbeitung von mehreren Haupt-Applikationen im Vordergrund.

Während GEM nur bedingt multitaskingfähig ist (eine Hauptapplikation, mehrere Accessories im Hintergrund), arbeitet demgegenüber das X/GEM vollständig ohne Accessories. Man startet einfach all diejenigen Programme, mit denen man arbeiten möchte, und schaltet zwischen ihnen hin und her, indem das entsprechende Fenster mit der Maus angewählt oder im Programm-Menü auf der rechten Seite der Menüzeile ein Programm gewählt wird. Dies wird auf einem Apple Macintosh und dem Multifinder genauso gemacht.

Die X/GEM Version ist zum jetzigen Zeitpunkt (April 1989) noch nicht ganz fertiggestellt. Wenn sie aber auf dem Markt ist, werden die Fähigkeiten eines INTEL 80386- oder MC68030-Prozessors zusammen mit einem Multitasking-Betriebssystem (z.B. FlexOS von Digital Research) voll ausgeschöpft werden.

Einige ATARI ST Besitzer werden jetzt denken, daß dies nur ein Traum für sie bleiben wird. Wir können aber schon heute diesen Traum nahezu vollständig verwirklichen, und

zwar mit der GEM 1.X-Version und der Hilfe von Accessories. In Kapitel 6 wird dieses Konzept genau erklärt. Es liegt dann nur noch an Ihnen, den Software-Entwicklern, in diese aufregende Welt des Multitasking unter GEM einzusteigen. Jeder ernsthafte Software-Entwickler, der nicht nur vor sich her programmieren möchte, sondern zeigen will, was er mit Hilfe von GEM leisten kann, kommt nicht umhin, dieses universelle und ausgereifte Konzept zu benutzen. Portable GEM-Applikationen können in kürzester Zeit erstellt werden.

### 2.1.5 Zukünftige Versionen

X/GEM für FlexOS ist aber immer noch nicht das Ende der GEM-Entwicklung. Digital Research hat angekündigt, GEM auch für das neue Multitasking-Betriebssystem OS/2 von Microsoft, der mit dem Presentation Manager ausgeliefert wird, zu portieren. Es wird dann keinen GEM-Desktop mehr geben. Vielmehr wird man vom DOS-Fenster aus GEM-Programme starten können, die dann in einem Fenster des Presentation-Manager ablaufen. Vorhandene GEM-Applikationen müssen lediglich neu übersetzt bzw. gelinkt werden. Sobald diese Version verfügbar sein wird, werden wir unsere Beispielapplikation daran anpassen.

Nur sauber geschriebene GEM-Applikationen werden dann ohne Probleme portierbar sein. Auf der CeBit '89 konnte man schon GEMDRAW unter OS/2 laufen sehen. Die gesamte GEM-Applikation lief dort in einem Fenster des Presentation Managers. Man konnte dann auch OS/2-Programme starten und zwischen diesen und GEMDRAW mit einem Mausklick umschalten. Das Ergebnis: Unbegrenztes Multitasking ohne Speicherprobleme.

Damit aber noch nicht genug. Auch für das UNIX-Betriebssystem soll eine Portierung gemacht werden. Wann aber damit zu rechnen ist, steht in den Sternen.

Wird das wahr, was Digital Research verspricht, so können GEM-Applikationen wie WORDPLUS oder ADIMENS, die schon heute auf mehreren Rechnern und Betriebssystemen lauffähig sind (und sauber programmiert wurden!!), künftig auch unter UNIX und OS/2 ihre Arbeit verrichten. Eine einheitliche Benutzerschnittstelle auf verschieden Systemen käme letztlich jedem Anwender zugute. Er bräuchte dann nicht mehr umzulernen, wenn er im Büro an einem UNIX-Terminal und zu Hause z.B. an einem PC oder ATARI ST arbeitet. Der Software-Entwickler schließlich könnte sein Produkt auf verschiedensten Systemen ohne Mehraufwand vermarkten.

## 2.2 Struktur

Bevor auf die beiden wichtigsten Teile des GEM eingegangen wird, soll die Gesamtstruktur erläutert werden. Besonders für den Anfänger ist es wichtig sie zu verstehen, damit er sich ein Bild vom Zusammenspiel der einzelnen Komponenten machen kann.

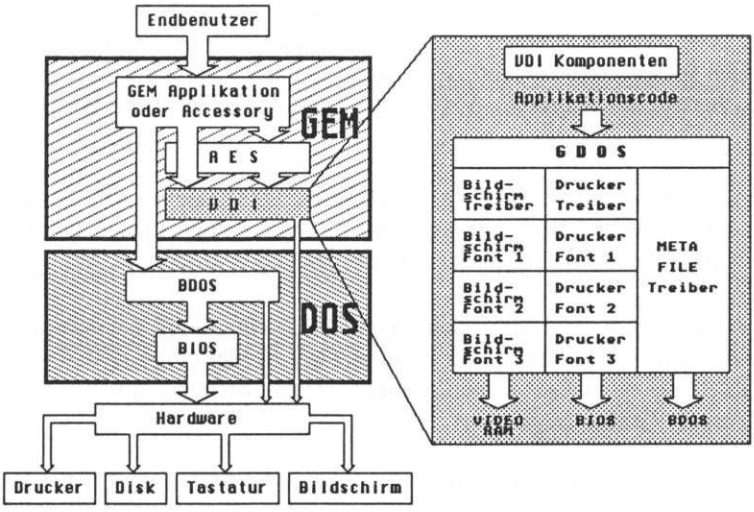



**Abb. 2.3:** GEM-Systemzusammenhang IBM

Ganz an unterster Stelle steht der Rechner mit seiner Hardware (Drucker, Harddisk/Floppy, Tastatur, Bildschirm, Maus etc). Um die Hardware zu steuern, bedarf es eines Betriebssystems. Dies ist bei einem IBM PC das sogenannte MS-DOS oder PC-DOS, auf einem ATARI ST das TOS (BIOS, XBIOS und GEMDOS).

Die Funktionen des Betriebsystems sind Speicherverwaltung, Steuern der Hardware (z.B. Bildschirmausgabe), Speicherung von Daten auf Massenspeicher, Laden und Starten von Programmen usw. Mit Hilfe des Betriebsystems ist es schon möglich, Software für einen Rechner zu betreiben. Dies kann eine Textverarbeitung, eine Datenbank oder z.B. eine Programmierumgebung wie Turbo Pascal sein. Da die Ausgabe solcher Software meist auf einem Bildschirm stattfindet, der nicht grafikfähig ist, nennen wir sie alphanumerische Software.

```
 Volume in Drive C is HARDDISK
 Directory of *.*

.              <DIR>   18-12-1987  17:30:06  10
..             <DIR>   00-00-1980  00:00:00  10
DOCS           <DIR>   13-05-1989  16:28:20  10
DRUCKER        <DIR>   13-05-1989  16:28:20  10
FORMATS        <DIR>   13-05-1989  16:28:22  10
IMAGES         <DIR>   13-05-1989  16:28:22  10
PRINTER  CFG     1779  19-12-1987  00:37:00  20
SPELLING DIC   176713  19-12-1987  00:37:10  20
TESTTEXT DOC     9101  19-12-1987  00:37:12  20
WORDHTRA HLP     4275  19-12-1987  00:37:16  20
WORDPLUS APP   158744  19-12-1987  00:41:46  20
WORDHELP RSC     1970  19-12-1987  00:37:36  20
WORDMSGS RSC     3986  19-12-1987  00:37:40  20
WORDPLUS RSC    22064  19-12-1987  00:37:42  20

 664000 bytes used in 17 file(s) and 1 subdirectory(s)

C:\WORDPLUS>
```

**Abb. 2.4:** Inhaltsverzeichnis einer Festplatte

Der Bildschirm wird bei alphanumerischer Software aufgeteilt in meist 80 Spalten zu 25 Zeilen. Jedes dieser 2000 Zeichen auf dem Schirm kann einzeln gesetzt werden. Dabei kann das Zeichen einen Wert zwischen 0 und 255 besitzen, was dem ASCII-Code gleichkommt. Das entsprechende Zeichen (z.B. 'A'=65) erscheint an der gewünschten Bildschirmposition.

Wie man sich leicht ausrechnen kann, sind die Möglichkeiten, 2000 Zeichen beliebig auf dem Bildschirm zu setzen, nicht gerade berauschend. Selbst mit sogenannten Grafiksymbolen sind allenfalls Tabellen und einfache Strichzeichnungen zu bewerkstelligen.

Um dieses Manko zu beheben, wurden Grafikkarten (IBM PC) eingeführt, bei denen jeder einzelne Punkt, aus dem ein Zeichen zusammengesetzt ist, an oder ausgeschaltet werden kann. Ein ATARI ST besitzt erst gar keinen speziellen Textbildschirm, er wird gleich mit einer hochauflösenden Grafik ausgeliefert.

Besteht ein Zeichen z.B. aus einer 8x16 Punktmatrix, so ergeben sich bei 80 Zeichen zu 25 Zeilen immerhin 640x400 Bildpunkte, die einzeln an oder ausgeschaltet werden können. Dies ist eine Steigerung der 2000 Zeichen aus dem Beispiel von oben auf 256000 Bildpunkte, also dem 128-fachen.

Während man mit einfachen Betriebssystemfunktionen noch Zeichen auf dem Bildschirm ausgeben konnte, wie z.B. printf ("Hallo"), so versagen diese Ausgabefunktionen bei einer Grafikkarte gänzlich, da das Betriebssystem bei seiner Entwicklung nicht vorhersehen konnte, welche Grafikkarten einmal für einen Rechner entwickelt werden würden.

Möchte man aber Software entwickeln, die diese Grafik benutzt, so muß entweder das Betriebssystem erweitert werden, oder man muß selbst die Grafikkarte über Hardwareadressen ansprechen und programmieren. Dieser Aufwand ist nicht nur unvertretbar hoch, auch Programme, die mit der einen Grafikkarte der Firma X laufen, versagen bei einer Karte der Firma Y den Dienst.

Dieses Problem haben Entwickler von Betriebsystemen erkannt und bieten zur ihren Systemen Grafikerweitwerungen an. Dabei handelt es sich um ein Softwarepaket, das resident in den Speicher geladen wird und welches über genormte Softwareschnittstellen angesprochen werden kann. Meistens sind dies Funktionen, die von einer höheren Proammiersprache aus (Pascal oder C) aufgerufen werden können.

Der Teil, der diese Grafikerweiterungen enthält, ist im GEM-System das VDI (Virtual Device Interface). Es enthält eine Menge von harwareunabhängigen Grafikfunktionen, die in jedes Programm mit eingebunden werden können. Da das VDI je nach Grafikkarte und Rechner bzw. Betriebssystem verschieden installiert wird und sich so der Hardware anpasst, erhöht die Verwendung dieser Funktionen in eigenen Programmen die Portabilität.

Moderne Software zeichnet sich aber nicht nur durch die Benutzung von hochauflösenden Rasterbildschirmen aus, sondern auch durch die Verwendung von Bildschirmfenstern,

Pull-Down-Menüs, Piktogrammen und einer Maus.

Die ganze Verwaltung der Bildschirmfenster, Menüs und Dialogboxen übernimmt ein zusätzlicher Teil des GEM, nämlich das AES (Application Environment Services). Es benutzt VDI-Funktionen, um z.B. ein Bildschirmfenster oder eine Dialogbox zu zeichnen oder ganze Fenster zu verschieben. Es setzt also größtenteils auf dem VDI auf.

Ein Anwendungsprogramm oder eine Applikation benutzt also AES-Funktionen, wenn es Fenster erzeugen, Dialogboxen benutzen oder Maus- und Tastatureingaben machen möchte. Für die geräteunabhängige Grafikausgabe wird man VDI-Funktionen benutzen, während man für die Speicherung und das Laden von Daten auf die gewohnten Betriebssystemfunktionen zurückgreift.

Die Vorteile liegen klar auf der Hand. Die riesige Sammlung von Grafik- und Fensterroutinen erleichtern die Entwicklung moderner Software. Der Programmierer muß sich nicht mehr mit hardwareabhängigen Ausgabefunktionen herumschlagen, sondern kann sich auf seine Aufgabe konzentrieren. Und die Software ist, vorausgesetzt er hält sich an die Schnittstellen des GEM-Systems, von einem Rechner auf den anderen portabel. Volle GEM-Portabilität von GEM 1.x bis X/GEM wird in Kapitel 6 erläutert.

## 2.3 VDI

Das Virtual Decive Interface (VDI) des GEM stellt dem Programmierer eine Menge von geräteunabhängigen Aus- und Eingabe-Funktionen zur Verfügung. Dabei spielt es normalerweise keine Rolle, ob z.B. Ausgaben auf den Bildschirm, den Drucker oder einen Plotter gemacht werden. Die Aufrufsequenzen sind praktisch identisch.

Leider bestehen, wie schon oben erwähnt, bei einem ATARI ST diese Möglichkeiten nicht, da die Teile des GDOS, die nicht für die Bildschirmausgabe bestimmt sind (Treiber und Zeichensätze für Plotter, Drucker, Metadateien, Kamera, Grafiktablett) keinen Platz mehr im ROM hatten. Da die GDOS-Erweiterungen zwar nachgeladen werden können (GDOS.PRG im Auto-Ordner), aber standardmäßig nicht mitgeliefert werden, konnte sich das GEM-Ausgabeformat (Bit-Image- und Metadatei-Format) noch nicht auf dem ATARI ST durchsetzen. Inzwischen gibt es aber glücklicherweise ein neues GDOS, das von Arndt Beißner entwickelt wurde und bei ATARI Deutschland GmbH erhältlich ist. Es zeichnet sich vor allem durch eine hohe Arbeitsgeschwindigkeit aus, während das alte Original-ATARI-GDOS den Bildschirmaufbau wesentlich bremste.

### 2.3.1 Koordinatensysteme

Grundsätzlich bietet das VDI 2 Möglichkeiten an, Grafik auf einem Ausgabegerät zu erzeugen.

Durch Benutzung von

a) Normalized Device Coordinates (NDC)
b) Raster Coordinates (RC)

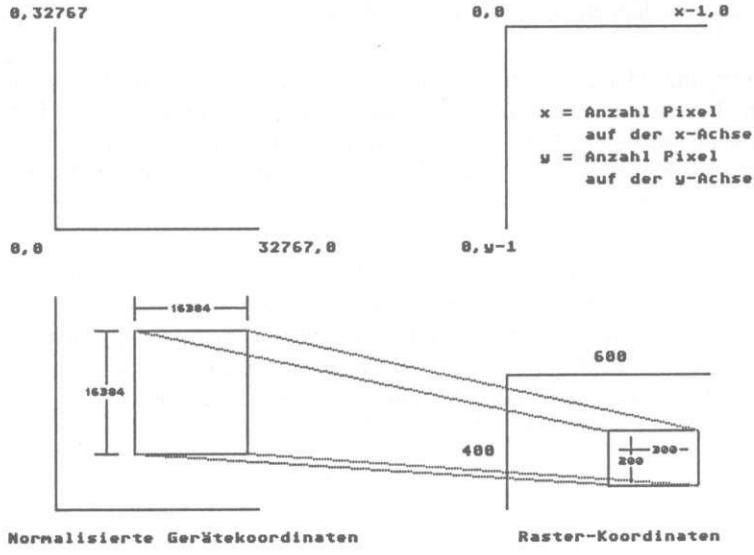



**Abb. 2.5:** NDC und RC-Koordinatensysteme

Die normalisierten Gerätekoordinaten (NDC) zeichnen sich dadurch aus, daß der Koordinatenursprung in der linken unteren Ecke liegt (0,0) und auf beiden Achsen die x- bzw. y-Werte bis 32767 reichen können.

Rasterkoordinaten (RC) haben ihren Ursprung in der linken oberen Ecke, und die Koordinaten wachsen nach rechts bzw. nach unten. Rasterkoordinaten sind bei einem Bildschirm z.B. 640 × 400 Punkte in x- bzw. y-Richtung. Für die Ausgabe in einem Bildschirmfenster wird man immer Rasterkoordinaten wählen, da das Fensterinnere seinen Ursprung in der linken oberen Ecke besitzt und nach rechts unten größer wird.

Ein Quadrat mit der Kantenlänge 16384 in NDC-Koordinaten wäre auf einem Bildschirm von 640 × 400 Punkten ein Rechteck mit 300 × 200 Punkten.
Das bedeutet, daß das Ausgabegerät als quadratisch angesehen wird. Ist es nicht physikalisch quadratisch, wie in unserem Bildschirmbeispiel, so übernimmt das GDOS die Transformation von NDC- in RC-Koordinaten, bevor die Ausgabe an den Treiber geht, der die Grafik auf dem Bildschirm darstellt.

Benutzt man nur Raster-Koordinaten, so wird vom GDOS keine Transformation vorgenommen. Die Ausgabe der Rasterpunkte (auch eine Linie besteht nur aus einzelnen Punkten) erfolgt direkt auf dem Ausgabegerät. Diese Methode ist wesentlich schneller und sollte der NDC-Methode vorgezogen werden.

### 2.3.2 Aufbau des VDI

Das VDI besteht aus 3 logischen Komponenten:

- Graphics Device Operatingsystem (GDOS)
- Gerätetreiber
- Zeichensätze

a) Graphics Device Operating System

Das GDOS beinhaltet alle geräteunabhängigen Grafikfunktionen. Diese Funktionen können z.B. über Aufrufe in C, Pascal oder Modula benutzt, werden. Falls normalisierte Koordinaten (NDC) benutzt werden (siehe 2.3.1), übernimmt das GDOS die Umrechnung in die gerätespezifischen Rasterkoordinaten.

b) Gerätetreiber

Ein Gerätetreiber ist ein Programm, das aus einer gegebenen Eingabe (z.B. „Zeichne Linie von Punkt 3,3 nach 50,30") ein physikalisches Gerät wie Bildschirm oder Drucker anspricht und dort die Eingabe sichtbar macht. Dabei kennt der Treiber die physikalischen Gegebenheiten eines Geräts und kann es optimal ansteuern.

So kann auf einem Laserdrucker die geforderte Linie mit einer Auflösung von $300 \times 300$ dpi (dots per inch = Punkte pro Zoll) dargestellt werden, während der Treiber eines Matrixdruckers nur eine Auflösung von vielleicht $240 \times 216$ dpi zu Papier bringen kann. Darauf beruht die Idee des VDI, die Qualität eines Ausgabegerätes durch entsprechende Treiber voll auszunutzen. Dies ist in GEM aber nur möglich, wenn Ausgaben über das sogenannte OUTPUT-Programm gemacht werden, welches ASCII-Texte, Metadateien und Bit-Image-Dateien mit der bestmöglichen Qualität zu Papier bringen kann.

c) Zeichensätze

Zu jedem Ausgabegerät gibt es noch eine Anzahl von Zeichensätzen, die installiert werden können. Die Zeichensätze sind so aufgebaut, daß das Verhältnis von Grafik- und Textausgaben auf allen Geräten konstant ist. Sie befinden sich als externe Dateien (z.B. IBMHSS36.FNT) auf dem Massenspeicher.

Der Treiber für den Bildschirm enthält standardmäßig einen sogenannten System-Zeichensatz (Nummer 1), der immer vorhanden ist. Er wird auch benutzt, um z.B. die Menüzeile anzuzeigen. Dieser Zeichensatz ist meist in 2 (IBM-PC) oder 3 (ATARI ST) Punktgrößen vorhanden. Die kleinste Größe ist u.a. für Piktogrammbeschriftungen vorgesehen.

Zusätzliche Zeichensätze können über eine VDI-Funktion jederzeit nachgeladen werden. Bei einem ATARI ST ist dies nur möglich, wenn das Programm GDOS.PRG installiert wurde. Welche und wieviele Zeichensätze zu einem Treiber gehören, kann man der Datei ASSIGN.SYS entnehmen. Sie wird weiter unten erklärt.

### 2.3.3 VDI-Funktionsgruppen

Bevor übersichtsartig auf die Funktionsgruppen eingegangen wird, sollen noch einige Begriffe erläutert werden:

- Arbeitsstation (workstation)
- Gerätekennung (device handle)
- Attribute

Eine Arbeitsstation ist ein allgemeiner Begriff für ein Ein- oder Ausgabegerät. Wie schon weiter oben erwähnt, können Arbeitsstationen Bildschirme, Drucker, Plotter, Kameras, Grafiktabletts usw. sein. Eine spezielle Art der Arbeitsstation sind die sogenannten Metadateien. Sie sind zwar keine physikalischen Geräte, werden aber genauso gehandhabt. Die Grafikausgabe wird in diesem Fall auf eine Datei (mit Suffix .GEM) statt z.B. auf einen Drucker geleitet. So kann diese Datei auch als Eingabe zum Datenaustausch dienen.

Das VDI erlaubt es, daß mehr als eine Arbeitsstation gleichzeitig von einem Programm genutzt wird. So kann man Ausgaben auf einen Plotter leiten und die Tastatur abfragen, ob die Ausgabe unterbrochen werden soll. Die Ausgabe geht dann auf die Arbeitsstation Plotter, während die Eingabe von den Arbeitsstationen Bildschirm, Tastatur oder Maus kommt. Die Arbeitsstationen Bildschirm, Tastatur und Maus werden immer als eine Einheit gesehen.

Es können auch gleichzeitig Ausgaben auf mehreren Geräten stattfinden. Die Funktionsaufrufe sind aber die gleichen (wie v_gtext = vdi graphics text). Wird z.B. Text auf Drucker und Bildschirm ausgegeben, so muß das VDI wissen, welches Gerät gemeint ist. Zu diesem Zweck übergibt man der Aus- oder Eingabefunktion eine Gerätekennung (handle). Es handelt sich um eine Zahl größer gleich 1. Damit entscheidet letztendlich das GDOS, zu welcher Arbeitsstation die gewünschte Funktion geleitet werden soll. Eine Gerätekennung erhält man beim Anmelden einer Arbeitsstation. Die Funktion v_opnwk (vdi open workstation) ist in diesem Zusammenhang entscheidend.

Jedes Gerät kennt 2 Sätze von Attributen. Die gerätespezifischen Attribute teilen dem Benutzer mit, wie das Aus- oder Eingabegerät beschaffen ist und welche Eigenschaften es hat. Zu diesen Eigenschaften zählen u.a. Anzahl der verfügbaren Farben, Pixelgröße sowie Breite und Höhe des Ausgabebereiches. Diese Attribute sind für jedes Ausgabegerät fest und können natürlich nicht verändert werden.

Außerdem existieren noch die sogenannten Ausgabe-Attribute. Mit ihnen kann u.a. die Strichdicke, die Farbe von Mustern und der Zeichensatz eingestellt werden, mit dem die Ausgabe erfolgen soll. Diese Attribute lassen sich dynamisch zur Laufzeit ändern.

Die Funktionen des VDI kann man in 7 Gruppen aufteilen:

- Kontroll-Funktionen

Mit Hilfe der Open-Workstation-Funktion können Ausgabegeräte zum späteren Gebrauch geöffnet werden. Man bekommt die Gerätekennung (device handle) zurück, mit der man alle nachfolgenden VDI-Aufrufe für dieses Gerät tätigen muß. Das GDOS kann dann den entsprechenden Ausgabetreiber aufrufen.

Für den Bildschirm gelten besondere Regeln. Es ist erlaubt, daß mehrere Applikationen gleichzeitig auf das physikalische Medium Bildschirm zugreifen. Beim Initialisieren des GEM übernimmt es selbst die Öffnung des physikalischen Ausgabegerätes Bildschirm mit der Funktion Open-Workstation (Gerätekennung 1). Jede GEM-Applikation hat dann die Möglichkeit, den Bildschirm als virtuelles Ausgabegerät zu öffnen. Dabei bekommt jede GEM-Applikation (Programm oder Desk Accessory) über seine virtuelle Gerätekennung einen Satz von Bildschirmattributen zugeordnet, die man individuell einstellen kann.

Wählt z.B. ein Programm die Farbe Rot für seine Textausgabe, ein Accessory aber die Farbe Grün und Kursivschrift, so sind in jeder der beiden Umgebungen die entsprechenden Attribute gesetzt. Es können Ausgaben vom Programm oder Accessory folgen, wobei das VDI die Texte entweder rot oder grün und kursiv auf den Bildschirm bringt. Die beiden Applikationen brauchen die Ausgabeattribute also nur zu Beginn des Programmlaufs einstellen. Sie bleiben individuell erhalten.

Beim Öffnen der Arbeitsstation wird außerdem der entsprechende Treiber geladen, der für die Ausgabe auf dem angegebenen physikalischen Medium zuständig ist. Den Namen des Treibers erfährt das System aus der Datei ASSIGN.SYS (bis GEM 2.X) bzw. aus dem Ordner \GEMAPPS\GEMSYS\ (ab GEM/3).

Weitere Funktionen sind für das Löschen und Aktualisieren der Arbeitsstation sowie für das Laden und Freigeben von Zeichensätzen zuständig.

– Ausgabe-Funktionen

Sie gehören zu den am meisten verwendeten Funtionen, da Applikationen ohne Ausgabe sehr selten einem Zweck dienen. Zu den Ausgabefunktionen gehören Linien, Streckenzüge, Text, gefüllte Streckenzüge, Rechtecke, Kreise, Kuchenstücke (pie slice), Ellipsen, Kreisbögen, ausgerichtete Texte und mehr.

– Attribute-Funktionen

Bei den Ausgabefunktionen ist es möglich, gewisse Attribute auszuwählen. So können bei einer Linie die Strichdicke, die Farbe, die Enden der Linie (z.B. ein Pfeil auf einer Seite) usw. eingestellt werden. Bei gefüllten Flächen können u.a. das Füllmuster, die Füllfarbe und weitere Attribute eingestellt werden. Wie oben erwähnt, behalten die Attribute für ein und dieselbe Arbeitsstation ihre Gültigkeit, bis das Programm terminiert oder die Attribute anders eingestellt werden.

– Raster-Funktionen

Mit Hilfe der Raster-Funktionen können rechteckige Blöcke von Punkten manipuliert werden. So können z.B. ganze Blöcke kopiert und vom Speicher in den Bildschirm übertragen werden. Dabei können logische Operationen bei der Anwendung benutzt werden. Es können auch Bildschirmbereiche gescrollt werden, wenn als Quelle und Ziel der Operation der Bildschirm gewählt wird. Dies ist eine der wichtigsten Anwendungen, wenn man an Bildschirmfenster denkt, die z.B. bei einer Textverarbeitung benutzt werden.

Raster-Funktionen haben aber noch eine andere wichtige Bedeutung, die bei ATARI ST-Programmierern häufig vernachlässigt wird. Es können auch Pixel-Blöcke vom Standardformat in das gerätespezifische Format umgewandelt werden. Versucht man ein ATARI ST GEM-Programm auf einen IBM-PC zu portieren, so wird man sich wundern, wenn z.B. Piktogramme gezeichnet werden sollen. Sie erscheinen je nach Grafikkarte mehr oder weniger verstümmelt. Die Bedeutung und Wirkungsweise der Transformation wird bei der Funktion TRANSFORM FORM in Kapitel 2.3.7 genau erklärt.

– Eingabe-Funktionen

Die Eingabe-Funktionen erlauben die Eingabe mit der Maus, den Funktionstasten und von Zeichen der Tastatur. Allerdings muß ausdrücklich darauf hingewiesen werden, daß diese Funktionen in GEM-Programmen nicht benutzt werden sollten oder besser dürfen. Nur in reinen VDI-Programmen funktionieren sie so, wie sie eigentlich sollen. Da sie für GEM-Programme also nicht wichtig sind, sollen sie auch nicht ausführlich behandelt werden.

Nur soviel sei gesagt: Man kann die Funktionen im Sample- und Request-Modus aufrufen. Im Sample-Modus kehrt die Funktion sofort zurück und gibt den Status und eventuell Daten an den Aufrufer zurück. Im Request-Modus wartet die Funktion so lange, bis ein Ereignis aufgetreten ist (z.B. eine Zeichenkette eingegeben wurde).

– Nachfrage-Funktionen

Mit ihnen ist es möglich, die eingestellten Attribute aus den Attribute-Funktionen zu erfragen. Auch Informationen über Zeichensätze werden angeliefert. Die erweiterte Abfrage-Funktion vq_extnd liefert weitere Informationen über das gewünschte Ausgabegerät.

– Escape-Funktionen

Einige spezielle Eigenschaften eines Ausgabegerätes können hiermit gesteuert werden. Unter anderem kann eine Hardcopy des Bildschirms auf ein Gerät ausgegeben werden. Auch die Steuerung des alphanumerischen Bildschirms erfolgt über Escape-Funktionen.

Bei einem ATARI ST ist der alphanumerische Bildschirm gleich dem Grafikschirm, bei IBM PC's und Kompatiblen existieren meist 2 getrennte Bildschirme: Der, den man beim Einschalten des Rechners zu sehen bekommt (DOS-Prompt), und derjenige, der beim Start von GEM und anderen grafikorientierten Programmen erscheint.

### 2.3.4 ASSIGN.SYS

Wie schon oben erwähnt, findet man bis GEM 2.X eine Datei namens ASSIGN.SYS. Sie wird dazu benötigt, Gerätetreiber und deren Zeichensätze festzulegen. Beim Installieren des GEM auf einem IBM PC werden die einzelnen Geräte, die angeschlossen werden sollen, abgefragt und die Datei ASSIGN.SYS entsprechend aufgebaut. Auf einem ATARI ST wird diese Datei von verschiedenen Desktop-Publishing-Systemen und objektorientierten Zeichenprogrammen mitgeliefert.

Wird das GEM gestartet, so untersucht das GDOS, das sich bei einem ATARI ST meist im AUTO-Ordner befindet, die Datei ASSIGN.SYS und baut im Speicher eine Zuweisungstabelle auf. Zu jeder angegebenen Gerätenummer wird der Name und die eindeutige Gerätenummer des Treibers und die Namen der für dieses Gerät verfügbaren zusätzlichen Zeichensätze in einer Tabelle gespeichert.

Öffnet man ein Ausgabegerät mit einer der verfügbaren Gerätenummern (über die Funktion v_opnwk), so referiert das GDOS die Zuweisungstabelle und lädt den Treiber für dieses Ausgabegerät in den Hauptspeicher. Der Treiber liegt nun vor, und alle VDI-Aufrufe mit der Gerätekennung (device handle) für diesen Treiber werden von ihm auf das Ausgabegerät gebracht.

Entsprechendes gilt für den VDI-Befehl vst_load_fonts. Das GDOS schaut in der Tabelle nach und lädt alle Zeichensätze für das angegebene Ausgabegerät.

Die Datei ASSIGN.SYS ist eine Textdatei und kann somit leicht eingesehen bzw. editiert werden. Ihr Aufbau wird im Folgenden beschrieben:

<Gerätenummer 1> <Option> <Dateiname des Treibers> <Kommentar>
<Zeichensatz 1>
<Zeichensatz 2>
...
<Gerätenummer 2> ...

**usw.**

– Gerätenummern

Die Gerätenummern hängen eng mit dem Gerätetyp zusammen. So sind für jeden Typ 10 Nummern vorgesehen.

| Gerätenummer | Gerätetyp |
|---|---|
| 01 – 10 | Bildschirm |
| 11 – 20 | Plotter |
| 21 – 30 | Drucker |
| 31 – 40 | Metadatei |
| 41 – 50 | Kamera |
| 51 – 60 | Grafiktablett |

Hinter der Gerätenummer können noch 2 Optionen angegeben werden. Ein 'r' bedeutet, daß der Treiber schon beim Untersuchen der Datei ASSIGN.SYS durch das GDOS geladen wird und resident im Speicher bleibt. Öffnet man die Arbeitsstation, so braucht der Treiber nicht erst nachgeladen werden. Allerdings kostet das einiges an Speicher. Ein einziger Treiber verbraucht zwischen 5 KByte (Metadatei) und 70 KByte (Lasertreiber).

Auf dem ATARI ST kann die Option noch den Buchstaben 'p' annehmen. Dies bedeutet, daß sich der Treiber permanent im ROM befindet, also nicht geladen werden muß. Dies ist für den Bildschirmtreiber der Fall.

Die Dateinamen der Treiber müssen mit einem Buchstaben beginnen und enden mit dem Suffix „.SYS". Kommentare werden mit einem Semikolon eingeleitet.

*Beispiele:*

```
01p SCREEN.SYS; Bildschirmtreiber, der sich im ROM befindet
21 FX80.SYS   ; Druckertreiber (21) mit Dateinamen FX80.SYS
31r META.SYS  ; Metadatei-Treiber, der resident im Speicher bleibt
```

Zwei Beispiele sollen den Gesamtaufbau verdeutlichen:

a) ASSISGN.SYS auf einem IBM PC mit Hercules-Grafikkarte

```
01 HERMONP6; Hercules Karte / Monochrom-PC-Bildschirm (720x348)
; Serielle Maus von Microsoft ( RS232 )
; Kommunikationsschnittstelle Nr. 1 (COM1:)
  IBMHSS07.FNT; Hercules Swiss  7 Punkt
  IBMHSS10.FNT; Hercules Swiss 10 Punkt
  IBMHSS14.FNT; Hercules Swiss 14 Punkt
  IBMHSS18.FNT; Hercules Swiss 18 Punkt
  IBMHSS36.FNT; Hercules Swiss 36 Punkt
  IBMHTR07.FNT; Hercules Dutch  7 Punkt
  IBMHTR10.FNT; Hercules Dutch 10 Punkt
  IBMHTR14.FNT; Hercules Dutch 14 Punkt
  IBMHTR18.FNT; Hercules Dutch 18 Punkt
  IBMHTR36.FNT; Hercules Dutch 36 Punkt
```

```
21 EPSMONH6; Epson Matrix Drucker (120 x 144 Punkte/Zoll)
; Parallelschnittstelle Nr. 1 für Drucker ( LPT1: )
  EPSHSS07.FNT; EPSON/IBM Swiss  7 Punkt (120x144 Punkte/Zoll)
  EPSHSS10.FNT; EPSON/IBM Swiss 10 Punkt (120x144 Punkte/Zoll)
  EPSHSS14.FNT; EPSON/IBM Swiss 14 Punkt (120x144 Punkte/Zoll)
  EPSHSS20.FNT; EPSON/IBM Swiss 20 Punkt (120x144 Punkte/Zoll)
  EPSHSS28.FNT; EPSON/IBM Swiss 28 Punkt (120x144 Punkte/Zoll)
  EPSHSS36.FNT; EPSON/IBM Swiss 36 Punkt (120x144 Punkte/Zoll)
  EPSHTR07.FNT; EPSON/IBM Dutch  7 Punkt (120x144 Punkte/Zoll)
  EPSHTR10.FNT; EPSON/IBM Dutch 10 Punkt (120x144 Punkte/Zoll)
  EPSHTR14.FNT; EPSON/IBM Dutch 14 Punkt (120x144 Punkte/Zoll)
  EPSHTR20.FNT; EPSON/IBM Dutch 20 Punkt (120x144 Punkte/Zoll)
  EPSHTR28.FNT; EPSON/IBM Dutch 28 Punkt (120x144 Punkte/Zoll)
  EPSHTR36.FNT; EPSON/IBM Dutch 36 Punkt (120x144 Punkte/Zoll)
31 METAFIL6; GEM Datei
```

Das GEM-System der o.g. Datei ASSIG.SYS wird mit 3 Ausgabegeräten betrieben:

1. Die Ausgabe auf den Bildschirm (Gerätekennung 01) erfolgt mit dem Bildschirmtreiber namens HERMONP6. Außer dem Systemzeichensatz (Menüs, Dialogboxen und Piktogrammbeschriftungen) stehen 2 weitere Zeichensätze in je 5 Punktgrößen zur Verfügung. Es handelt sich um die Sätze Swiss und Dutch (Times Roman) mit den Punktgrößen 7, 10, 14, 18 und 36 Punkt.

2. Das zweite Ausgabegerät (Gerätekennung 21) ist ein Matrixdrucker der Standard-Epson-Serie. Der Treiber heißt EPSMONH6 und ist in der Lage, eine Grafikauflösung von maximal $120 \times 144$ Punkte/Zoll zu erreichen. Für diesen Treiber stehen die beiden Zeichensätze Swiss und Dutch mit den Punktgrößen 7, 10, 14, 20 28 und 36 zur Verfügung.

3. Als drittes Ausgabegerät (Gerätekennung 31) kann der Metadatei-Treiber METAFIL6 benutzt werden. Alle Grafikausgaben über ihn werden auf eine Datei geschrieben, die z.B. von GEMDRAW oder anderen Grafikpaketen wieder eingelesen werden kann. Das Vorhandensein dieses Treibers ist entscheidend für den grafischen Datenaustausch zwischen GEM-Programmen.

b) ASSIGN.SYS auf einem ATARI ST

```
path = c:\gemsys\
01p screen.sys
IBMHSS10.FNT
IBMHSS14.FNT
IBMHSS18.FNT
IBMHSS36.FNT
02p screen.sys
LOWRES10.FNT
```

```
LOWRES14.FNT
LOWRES18.FNT
LOWRES36.FNT
03p screen.sys
MEDRES10.FNT
MEDRES14.FNT
MEDRES18.FNT
MEDRES36.FNT
04p screen.sys
HIRES10.FNT
HIRES14.FNT
HIRES18.FNT
HIRES36.FNT
21 FX80.SYS
EPSHSS10.FNT
EPSHSS14.FNT
EPSHSS20.FNT
EPSHSS28.FNT
EPSHSS36.FNT
31 META.SYS
```

Auf einem ATARI ST gilt für die Datei ASSIGN.SYS im Prinzip das gleiche wie für einen IBM-PC. Allerdings gibt es hier noch einige Besonderheiten:

Die erste Zeile der Datei kann einen Pfad, also Laufwerksangabe und Ordnerhierarchie beinhalten. Er gibt an, wo sich die Treiber und Zeichensätze befinden sollen, die nachgeladen werden können.

Wie man an obigem Beispiel sieht, gibt es vier Ausgabegeräte für den Bildschirm, deren Kennungen von 1 – 4 reichen. Es sind aber keine vier echten Ausgaberäte gemeint, da in allen Fällen der Bildschirmtreiber SCREEN.SYS benutzt wird, der sich permanent ('p') im ROM befindet und den ATARI-Bildschirm betreibt. Gerät Nummer 01, also der Bildschirm, ist das Default-Ausgabegerät. Das GEM öffnet mit dieser Nummer ein physikalisches Ausgabegerät, bevor der Desktop als erste Applikation gestartet wird. Dies ist auch die Nummer, die bei dem AES-Aufruf „graf_handle" zurückgegeben wird.

Die Nummern 2 – 4 beziehen sich auf die verschiedenen Auflösungen, die ein ATARI ST annehmen kann. Dabei bedeuten

02 = niedrige Auflösung (low resolution, 320 × 200, 16 Farben)
03 = mittlere Auflösung (medium resolution, 640 × 200, 4 Farben)
04 = hohe Auflösung (high resolution, 640 × 400, monochrom)

Weitere Nummern werden wohl bei neueren Rechnern, wie beim ATARI TT, eine Rolle spielen.

Wozu werden diese Nummern eigentlich benötigt? Im Grunde käme man mit der Gerätekennung 01 aus. Lädt man Zeichensätze nach, so werden die entsprechenden Zeichensätze für alle drei Auflösungen benutzt. Wenn man sich aber einmal die mittlere Auflösung eines ATARI ST angesehen hat, so stellt man fest, daß das Verhältnis von x:y gleich 1:2 ist, also y = 2x ist. Alle Punkte in y-Richtung sind um die doppelte Länge gedehnt. Dies sieht man sofort an den Piktogrammen des Desktop, die in der Länge verzerrt erscheinen.

Um ein Auflösungsverhältnis von 1:1 erreichen, was bei hoher und niedriger Auflösung der Fall ist, müßte man entweder alle Punkte in x-Richtung verdoppeln oder alle Punkte in y-Richtung halbieren. Auch Zeichensätze sind nur eine Aneinanderreihung von einzelnen Punkten. Wurden sie für die hohe Auflösung entworfen, so erscheinen sie in der mittleren Auflösung genauso verzerrt wie Piktogramme. Speziell für diese mittlere Auflösung sollte man also getrennte Zeichensätze entwerfen. Um diese dann auch laden zu können, öffnet man eine virtuelle Arbeitsstation mit der Nummer 03.

Um also immer in jeder Auflösung die korrekten Zeichensätze parat zu haben, muß mittels folgender Formel die Gerätekennung in Abhängigkeit der Auflösung ermittelt werden:

```
device_id = Getrez () + 2;
```

Als Gerätekennung erhält man 2, 3 oder 4 je nach Auflösung.

c) Treiberinformationen ab GEM/3

Ab Version GEM/3 existiert die Datei ASSIGN.SYS nicht mehr. Dort werden die Treiber der Geräte, die installiert werden sollen, in den Ordner \GEMAPPS\GEMSYS kopiert.

Die Zeichensätze der Ausgabegeräte werden bei der Installation in den Ordner \GEMAPPS\FONTS kopiert.

Wie kann aber ein Anwendungsprogramm bestimmen, welche Gerätetreiber und Zeichensätze zur verfügung stehen? Dazu wird im Programmer's Toolkit eine neue VDI-Funktion zur Verfügung gestellt, sie heißt:

```
v_get_driver_info (device_id, info_select, info_string)
```

Beim Aufruf bekommt man zu jeder Gerätenummer (s.o.) eine Zeichenkette, die Informationen darstellt. Diese Informationen werden durch den Parameter info_select gesteuert. Er kann Werte von 1 bis 5 annehmen. Je nach info_select werden im info_string folgende Informationen zurückgegeben:

| info_select | info_string |
|---|---|
| 1 | Dateinamen des Treiber |
| 2 | Bezeichnung des Gerätes |
| 3 | Zusatzinformationen, falls vorhanden |
| 4 | Zeichensatzdateien eines Gerätes |
| 5 | Patchadresse des Treibers (z.B. Drucker-Port-Nummer) |

Wir schlagen für info_select folgende Konstanten vor:

```
#define DRV_FNAME  1
#define DRV_NAMES  2
#define DRV_XNAMES 3
#define DRV_FONTS  4
#define DRV_PATCH  5
```

Das nachfolgende C-Programm listet alle Informationen zu allen verfügbaren Geräten auf:

```
#include <stdio.h>
#include <portab.h>
#include <vdi.h>

/****** Variablen ***************************************************/

GLOBAL WORD contrl [12];
GLOBAL WORD intin [128];
GLOBAL WORD ptsin [128];
GLOBAL WORD intout [128];
GLOBAL WORD ptsout [128];

/****** Prototypen **************************************************/

EXTERN VOID vdi                _((VOID));
LOCAL  VOID v_get_driver_info  _((WORD device_id, WORD info_select,
                                  UBYTE *info_string));

/****** Funktionen **************************************************/

LOCAL VOID v_get_driver_info (device_id, info_select, info_string)
WORD      device_id;
WORD      info_select;
UBYTE FAR *info_string;

{
   WORD i;
   BYTE *bptr;

   contrl [0] = -1;
   contrl [1] = 0;
   contrl [3] = 2;
   contrl [5] = 4;
   contrl [6] = 0;

   intin [0] = device_id;
   intin [1] = info_select;
```

```
    vdi ();

    if (info_select != DRV-PATCH)
    {
       bptr = (BYTE *)intout;
       for (i = 0; i < contrl [4]; i++) *info_string++ = *bptr++;
       *info_string = 0;
    } /* if */
    else
       *(WORD *)info_string = intout [0];
} /* v_get_driver_info */

/*****************************************************************/

GLOBAL WORD main ()

{
  UBYTE info_string [255];
  WORD  device_id, info_select, device;

  for (info_select = DRV_FNAME; info_select <= DRV_PATCH; info_select++)
  {
    printf ("\ninfo_select: %d\n", info_select);

    for (device_id = 1; device_id <= 60; device_id += 10)
    {
      device = device_id;

      do
      {
        v_get_driver_info (device, info_select, info_string);

        if (info_select == DRV_PATCH)
            printf ("device id: %2d, info_word = %d\n",
                     device, *(WORD *)&info_string [0]);
          else
            if (*info_string)
              printf ("device id: %2d, info_string = %s\n",
                       device, info_string);

        device++;
      } while (*info_string && (device % 10 != 0));
    } /* for */
  } /* for */

  return (0);
} /* main */
/*****************************************************************/
```

Folgende Geräte wurden unter GEM/3 installiert:

a) eine EGA-Karte
b) ein HP-Plotter
c) ein Drucker der Marke EPSON LQ
d) ein Drucker der Marke EPSON FX
e) Treiber für Metadatei

Dann ist die Ausgabe des obigen Beispielprogrammes folgende:

```
info_select: 1
device id:  1, info_string = C:\GEMAPPS\GEMSYS\SDEHF9.EGA
device id: 11, info_string = C:\GEMAPPS\GEMSYS\VDHPX8.SYS
device id: 21, info_string = C:\GEMAPPS\GEMSYS\PDELQ9.ELQ
device id: 22, info_string = C:\GEMAPPS\GEMSYS\PDEHI9.EPS
device id: 31, info_string = C:\GEMAPPS\GEMSYS\MDGEM9.SYS

info_select: 2
device id:  1, info_string = EGA HiRes 16
device id: 11, info_string = HPGL Plotter
device id: 21, info_string = Epson LQ
device id: 22, info_string = IBM/Epson Hi

info_select: 3
device id: 11, info_string = Hewlett Packard, Bruning, Calcomp
                             und andere HP-Kompatible
device id: 21, info_string = Epson Drucker der LQ Serie
                             (180 x 180 Punkte/Inch)

info_select: 4
device id:  1, info_string = C:\GEMAPPS\FONTS\*.EGA
device id: 11, info_string = C:\GEMAPPS\FONTS\*.SYS
device id: 21, info_string = C:\GEMAPPS\FONTS\*.ELQ
device id: 22, info_string = C:\GEMAPPS\FONTS\*.EPS
device id: 31, info_string = C:\GEMAPPS\FONTS\*.SYS

info_select: 5
device id:  1, info_word = 768
device id: 11, info_word = 3
device id: 12, info_word = 3
device id: 13, info_word = 3
device id: 14, info_word = 3
device id: 15, info_word = 3
device id: 16, info_word = 3
device id: 17, info_word = 3
device id: 18, info_word = 3
device id: 19, info_word = 3
device id: 21, info_word = 1
device id: 22, info_word = 0
```

d) Zusammenfassung

Die Datei ASSIGN.SYS wird vom GEM-System benötigt, um festzustellen, welche Ausgabegeräte im System zur Verfügung stehen und von welchem Treiber sie bedient werden. Sie gibt außerdem an, welche Zeichensätze für ein Gerät zur Verfügung stehen. Ab GEM/3 werden die Ordner GEMSYS und FONTS benutzt.

Jedes Gerät wird über eine Gerätekennung (device handle) angesprochen. Öffnet man eine physikalische Arbeitsstation für dieses Gerät (v_opnwk), so wird der Gerätetreiber geladen und man kann Ausgaben tätigen.

Durch den VDI-Befehl vst_load_fonts werden automatisch die zusätzlichen Zeichensätze für das angegebene Gerät nachgeladen, und man bekommt die Anzahl der geladenen Zeichensätze zurück.

### 2.3.5 Kommunikation mit dem VDI

Alle GEM-Programme nutzen das VDI, dessen Funktionalität fast alles bietet, um Grafik auf ein Ausgabegerät zu bringen. Für Programme, die in einer höheren Programmiersprache geschrieben sind, erfolgt die Nutzung des VDI einfach durch den Aufruf über die Funktionsnamen.

Assembler-Programmierer haben es da schwerer. Sie müssen verschiedene Strukturen beachten, um die Parameter an das VDI zu liefern. Obwohl es kaum Sinn macht, VDI-Befehle in Assembler zu nutzen, soll die Parameterübergabe doch erklärt werden.

Dies hat zwei Gründe: zum einen dient es dem Verständnis des GEM-Systems, zum anderen benötigt derjenige diese Information, der sich näher mit Metadateien beschäftigen möchte.

Das VDI wird über ein einziges Unterprogramm aufgerufen, dem 5 Parameter übergeben werden. Es handelt sich um 5 Felder (arrays), wobei der Grundtyp eines Feldes aus einem Rechnerwort, also 16 Bit bestehen muß. Die Felder müssen folgende Namen tragen:

- contrl (Funktionsangabe, Gerätekennung usw.)
- intin (ganzzahlige Eingaben)
- ptsin (Feld für Eingabe von Koordinatenpunkten)
- intout (ganzzahlige Ausgaben)
- ptsout (Ausgaben von Koordinatenpunkten)

Die Größen der Arrays dürfen eine gewisse Länge nicht unterschreiten. Da man nie weiß, ob bei späteren GEM-Versionen (X/GEM) nicht mehr Feldelemente benötigt werden, werden die int- und pts-Arrays etwas großzügiger angelegt. In C-Schreibweise lautet dies:

```
GLOBAL WORD contrl [12];
GLOBAL WORD intin [256];
GLOBAL WORD ptsin [256];
GLOBAL WORD intout [256];
GLOBAL WORD ptsout [256];
```

GLOBAL bedeutet in diesem Zusammenhang, daß die Felder beim Linkvorgang nach außen hin sichtbar sind, WORD bedeutet 16 Bit-Werte. Die Bedeutung solcher Portabilitäts-Makros wird in Kapitel 3 geklärt.

– Der Parameter Block

Er hat nur Bedeutung, wenn man selbst die Parameter an das VDI übergeben möchte. Im Normalfall werden über High-Level-Funktionsaufrufe (C, Pascal usw.) die benötigten Parameter aufbereitet und an das VDI übergeben. Dies besorgen die sogenannten VDI- und AES-Bindings (sprich beindings).

Um die Parameter selbst an das VDI zu übergeben, muß der sogenannte Parameter-Block erstellt werden. Er hat folgenden einfachen Aufbau:

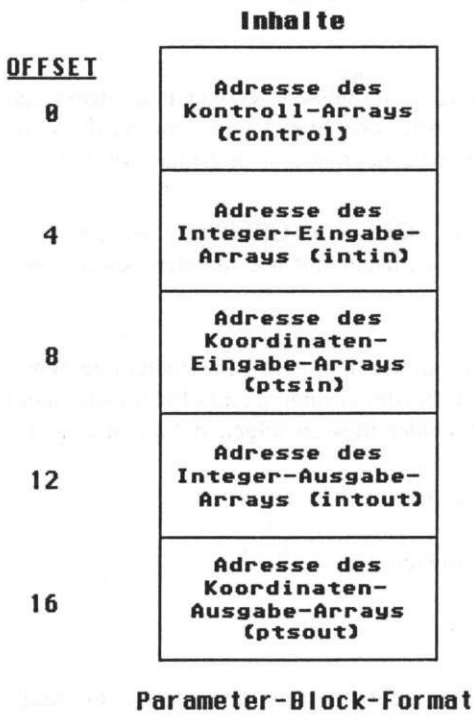



**Abb. 2.6:** VDI Parameter-Block

In C-Schreibweise könnte man diesen Block wie folgt beschreiben:

```
struct vdi_parameters
{
  WORD *contrl;
  WORD *intin;
  WORD *ptsin;
  WORD *intout;
  WORD *ptsout;
};

struct vdi_parameters pblock =
  {contrl, intin, ptsin, intout, ptsout};
```

Die Variable pblock enthält also die Adressen der 5 VDI-Parameterfelder. Nun muß nur noch die Adresse der Variablen pblock sowie eine ID-Nummer in die entsprechenden Register geladen werden. Anschließend erfolgt ein Software-Interrupt oder ein Trap-Befehl.

a) INTEL-Modell (IBM PC's und Kompatible)

Die Adresse des Parameter-Blockes muß in die Register DS:DX, sowie der ID-Wert 1139 (Hex $473) in das Register CX übertragen werden. Anschließend muß der Interrupt $EF ausgeführt werden.

In 8086-Assembler könnte dies wie folgt aussehen:

```
        extrn   _pblock:word

_vdi:   push    ds
        mov     dx, seg   _pblock
        mov     ds, dx
        mov     dx,offset _pblock ;ds:dx zeigt auf den parm block
        mov     cx,473h
        int     0EFh
        pop     ds
        ret
```

b) 68000-Modell (ATARI ST, TT)

Die Adresse des Parameter-Blockes muß in das Register D1.L, sowie der ID-Wert 115 (Hex $73) in das Register D0.W übertragen werden. Anschließend muß der Interrupt TRAP #2 ausgeführt werden.

In 68000-Assembler könnte dies wie folgt aussehen:

```
        .globl _pblock

_vdi:   move.l #_pblock,D1
        move.w #$73,D0
        trap   #2
        rts
```

Für beide Modelle gilt dann, daß das VDI angesprungen wird und die Parameter ausgewertet werden. Das VDI füllt die Ausgabeparameter intout und ptsout, da es ja die Adressen der beiden Felder kennt. Von dort kann dann die Applikation die Werte wieder übernehmen. Benutzt man eine höhere Programmiersprache, übernehmen die Bindings (VDI- und AES-Libraries) die Übertragung der Werte aus den Ausgabearrays in die Parameter der Funktionsaufrufe.

Da das Feld „contrl" bei den VDI-Aufrufen eine spezielle Bedeutung hat, soll der Aufbau kurz erklärt werden:

contrl [0] = Opcode (Nummer) der VDI-Funktion
contrl [1] = Anzahl der x,y-Punkte im Feld ptsin
contrl [2] = Anzahl der x,y-Punkte im Feld ptsout
contrl [3] = Länge des Feldes intin
contrl [4] = Länge des Feldes intout
contrl [5] = Nummer der Unterfunktion für GDP oder Escape-Funktionen
contrl [6] = Gerätekennung (device handle)
contrl [7..11] = weitere Informationen (Funktionsabhängig)

### 2.3.6 Ein erstes VDI-Beispiel

Nach so viel Theorie soll ein kurzes VDI-Programm vorgestellt werden. Das Programm befindet sich auf der Begleitdiskette unter dem Namen „VDITEST.C". Es zeichnet ein rotes Haus aus 5 Linien auf den Bildschirm. Auf die Grundlinie des Hauses wird der grüne Text „VDI-Test Haus" in der Punktgröße 10 geschrieben. An diesem Beispiel sollen alle VDI-typischen Aufrufsequenzen und ihre Wirkung erläutert werden.

Doch zuvor soll noch kurz auf die Möglichkeit eingegangen werden, wie man das GEM bzw. VDI auf einem PC unter MS-DOS starten kann.

Um ein reines VDI-Programm unter GEM 1.X bzw 2.X zu starten, also ohne Fenster und Menüs, muß „GEMVDI /PROGRAMMNAME" eingegeben werden. GDOS und ASSIGN.SYS müssen vorhanden sein. Ab GEM/3 muß GEMVDI -PROGRAMMNAME eingegeben werden.

Um normale GEM-Programme zu starten, gibt man „GEMVDI FILENAME“ ein. Der Desktop wird in diesem Fall nicht gestartet. Wird die gestartete Applikation beendet, kehrt man zum DOS-Prompt zurück.

Um den GEM Desktop zu starten, muß man nur

```
cd \GEMSYS bzw. cd \GEMAPPS\GEMSYS bei GEM/3
GEMVDI
```

eingeben. Nach der Installierung von GEM auf einem PC reicht auch die Eingabe von GEM. Es handelt sich um eine Batch-Datei (GEM.BAT), die vorher noch auf das Verzeichnis (\GEMAPPS)\GEMSYS umschaltet, bevor das GEMVDI aufgerufen wird.

Auf einem ATARI ST ist es nicht möglich, nur das VDI zu starten, da das GEM-System intern mittels GEMVDI gestartet wird und dann schon der Desktop als erste Applikation hochkommt. Diese Aufruffolge liegt in den ROMs der ST's fest.

Nun zu unserem ersten C-Beispielprogramm, das folgendes tut: Es soll ein rotes Haus zeichnen und am unteren Rand des Hauses einen grünen Text in der Punktgröße 10 ausgeben. Die Ausgabe soll zunächst im NDC-Koordinatensystem und dann im Rastersystem erfolgen. Außerdem soll die Ausgabe auf den Bildschirm, den Plotter, den Drucker und anschließend auf eine Datei gelenkt werden. Dateien, die solche Grafikbefehlsfolgen beinhalten, nennt man auch Metadateien. Sie werden weiter unten erklärt. Sowohl die Metadateien als auch die Bit-Image-Pixeldateien sind wichtigstes Hilfsmittel, um grafische Informationen zwischen beliebigen Programmen austauschen zu können.

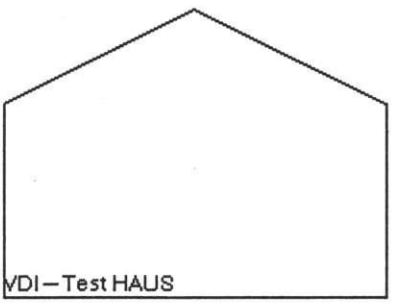



**Abb. 2.7:** Ausgabe des Programms VDITEST.C auf den Bildschirm

Das Beispiel als C-Programm sieht wie folgt aus:

```
/* VDI TEST-Programm                  */
/* Zeichnet ein Haus und einen Text */
/* auf ein beliebiges Ausgabegerät  */
```

```
#include <stdio.h>
#include <string.h>
#include <portab.h>
#include <vdi.h>
#include <aes.h>

#define SCREEN     01
#define PLOTTER    11
#define PRINTER    21
#define METAFILE   31
#define CAMERA     41
#define TABLET     51

#define NDC         0
#define RC          2

#define VM_PAGESIZE 0
#define VM_COORDS   1

#define SWISS       2 /* SWISS font */

#define abs(x)      ((x) <  0  ? -(x) : (x))  /* Absolut-Wert */
#define max(x,y)    (((x) > (y)) ? (x) : (y)) /* Maximim-Funktion */
#define min(x,y)    (((x) < (y)) ? (x) : (y)) /* Minimum Funktion */

/****** Variablen ***************************************************/

GLOBAL WORD   contrl [12];
GLOBAL WORD   intin [128];
GLOBAL WORD   ptsin [128];
GLOBAL WORD   intout [128];
GLOBAL WORD   ptsout [128];

LOCAL  WORD    work_in [103];
LOCAL  WORD    work_out [57];
LOCAL  WORD    vdi_handle;
LOCAL  BOOLEAN from_desktop;
LOCAL  BYTE    meta_name [80];
LOCAL  WORD    min_x, min_y, max_x, max_y;
LOCAL  WORD    screen_w, screen_h;
LOCAL  WORD    meta_w, meta_h;

/****** Prototypen **************************************************/

EXTERN VOID    vdi        _((VOID));
```

```
LOCAL  BOOLEAN open_work  _((WORD device, WORD coord));
LOCAL  VOID    close_work _((WORD device, WORD coord));
LOCAL  VOID    wait       _((VOID));
LOCAL  VOID    house      _((WORD device, WORD coord));

/*****************************************************************/

#if GEMDOS      /* sind ab GEM 2.X in den Bindings */
GLOBAL VOID vm_pagesize (vdi_handle, pgwidth, pgheight)
WORD vdi_handle, pgwidth, pgheight;

{
  contrl [0] = 5;
  contrl [1] = 0;
  contrl [3] = 3;
  contrl [5] = 99;
  contrl [6] = vdi_handle;

  intin [0] = VM_PAGESIZE;
  intin [1] = pgwidth;
  intin [2] = pgheight;

  vdi ();
} /* vm_pagesize */

/*****************************************************************/

GLOBAL VOID vm_coords (vdi_handle, llx, lly, urx, ury)
WORD vdi_handle, llx, lly, urx, ury;

{
  contrl [0] = 5;
  contrl [1] = 0;
  contrl [3] = 5;
  contrl [5] = 99;
  contrl [6] = vdi_handle;

  intin [0] = VM_COORDS;
  intin [1] = llx;
  intin [2] = lly;
  intin [3] = urx;
  intin [4] = ury;

  vdi ();
} /* vm_coords */
#endif

/*****************************************************************/
```

```
LOCAL BOOLEAN open_work (device, coord)
WORD device;
WORD coord;

{  WORD i;

  for (i = 0; i < 103; i++) work_in [i] = 1;

  work_in [0]  = device; /* device handle       */
  work_in [10] = coord;  /* NDC/RC Koordinaten  */

  if (device != SCREEN)  /* GEM/3 Erweiterungen */
  {
    work_in [11] = OW_NOCHANGE; /* Paralleler oder Serieller Port */
    work_in [12] = 0;           /* Port #0 */
  } /* if */

  if (device == SCREEN)
  {
    if (from_desktop)
    {
      appl_init ();
      vdi_handle = graf_handle (&i, &i, &i, &i);
      v_opnvwk (work_in, &vdi_handle, work_out); /* virt. öffnen */
      graf_mouse (M_OFF, NULL);
    }
    else
    {
      v_opnwk (work_in, &vdi_handle, work_out); /* phys. öffnen */
      v_hide_c (vdi_handle);
    } /* else */

    screen_w = work_out [0] + 1;
    screen_h = work_out [1] + 1;
  } /* if */
  else                                          /* nicht Bildschirm */
    v_opnwk (work_in, &vdi_handle, work_out);

  if (vdi_handle > 0)
  {
    switch (device)
    {
      case SCREEN   : v_clrwk (vdi_handle);                 break;
      case PLOTTER  : v_updwk (vdi_handle);                 break;
      case PRINTER  : v_updwk (vdi_handle);                 break;
      case METAFILE : vm_filename (vdi_handle, meta_name); break;
    } /* switch */
```

```
    vst_load_fonts (vdi_handle, 0);
  } /* if */

  return (vdi_handle > 0)
} /* open_work */

/*************************************************************************/

LOCAL VOID close_work (device, coord)
WORD device;
WORD coord;

{
  WORD llx, lly, urx, ury;

  if (device == SCREEN) wait ();

  if (from_desktop)
    graf_mouse (M_ON, NULL);
  else
    v_show_c (vdi_handle, TRUE);

  if (from_desktop && device == SCREEN)
  {
    vst_unload_fonts (vdi_handle, 0);
    v_clsvwk (vdi_handle);
    appl_exit ();
  } /* if */
  else
  {
    switch (device)
    {
      case SCREEN   : break;
      case PLOTTER  : v_updwk (vdi_handle); break;
      case PRINTER  : v_updwk (vdi_handle); break;
      case METAFILE : if (coord == NDC)
                      {
                        llx = 0;
                        lly = 0;
                        urx = meta_w;
                        ury = meta_h;
                      }
                      else /* RC */
                      {
                        llx = 0;
                        lly = meta_h;
```

```
                        urx = meta_w;
                        ury = 0;
                      } /* else */

                      v_meta_extents (vdi_handle, min_x, min_y,
                                                  max_x, max_y);
                      /* Letter size =  7,50 x 10.00 Zoll */
                      /*             = 19,05 x 25,40 cm   */
                      vm_pagesize (vdi_handle, 1905, 2540);
                      vm_coords (vdi_handle, llx, lly, urx, ury);
                      break;
    } /* switch */

    vst_unload_fonts (vdi_handle, 0);
    v_clswk (vdi_handle);
  } /* else */
} /* close_work */

/*********************************************************************/

LOCAL VOID wait ()

{
  WORD echo_xy [2];
  BYTE string [2];

  echo_xy [0] = 0;
  echo_xy [1] = 0;

  if (from_desktop)
    evnt_keybd ();
  else
    vrq_string (vdi_handle, 1, 0, echo_xy, string);
} /* wait */

/*********************************************************************/

LOCAL VOID house (device, coord)
WORD device;
WORD coord;

{
  WORD i, pxy [12];

  if (open_work (device, coord))
  {
```

```
if (coord == NDC)
{
  pxy [0]  = 6000;
  pxy [1]  = 6000;
  pxy [2]  = 6000;
  pxy [3]  = 10000;
  pxy [4]  = 12000;
  pxy [5]  = 12000;
  pxy [6]  = 14000;
  pxy [7]  = 10000;
  pxy [8]  = 14000;
  pxy [9]  = 6000;
  pxy [10] = 6000;
  pxy [11] = 6000;

  if (device == METAFILE)
  {
    min_x = pxy [0];
    min_y = pxy [1];
    max_x = pxy [6];
    max_y = pxy [5];

    meta_w = 32766;
    meta_h = 32766;
  } /* if */
} /* if */
else
{
  pxy [0]  = 100;
  pxy [1]  = 200;
  pxy [2]  = 100;
  pxy [3]  = 100;
  pxy [4]  = 200;
  pxy [5]  = 50;
  pxy [6]  = 300;
  pxy [7]  = 100;
  pxy [8]  = 300;
  pxy [9]  = 200;
  pxy [10] = 100;
  pxy [11] = 200;

  if (device == METAFILE)
  {
    min_x = pxy [0];
    min_y = pxy [5];
    max_x = pxy [6];
    max_y = pxy [1];
```

```
        meta_w = screen.w;
        meta_h = screen.h;
      } /* if */
    } /* else */

    vsl_color (vdi_handle, RED);
    vsl_type (vdi_handle, SOLID);
    v_pline (vdi_handle, 6, pxy); /* Haus zeichnen */

    vswr_mode (vdi_handle, MD_TRANS);
    vst_color (vdi_handle, GREEN);
    vst_font (vdi_handle, SWISS);
    vst_effects (vdi_handle, TXT_NORMAL);
    vst_alignment (vdi_handle, ALI_LEFT, ALI_BOTTOM, &i, &i);
    vst_point (vdi_handle, 10, &i, &i, &i, &i);
    vst_rotation (vdi_handle, 0);
    v_gtext (vdi_handle, pxy [0], pxy [1], "VDI-Test HAUS");

    close_work (device, coord);
  } /* if */
} /* house */

/*****************************************************************/

GLOBAL WORD main ()

{
  screen_w = 640; /* default Werte            */
  screen_h = 200; /* werden später korrigiert */

  from_desktop = TRUE; /* FALSE für MSDOS, wenn GEMVDI */
                       /* direkt gestartet wird        */

  house (SCREEN, NDC);
  house (SCREEN, RC);

  house (PLOTTER, NDC);
  house (PLOTTER, RC);

  house (PRINTER, NDC);
  house (PRINTER, RC);

  strcpy (meta_name, "HOUSENDC.GEM");
  house (METAFILE, NDC);

  strcpy (meta_name, "HOUSERC.GEM");
  house (METAFILE, RC);
```

```
  return (0);
} /* main */
```

Zur Schreibweise des Programms soll noch einiges vorweg gesagt werden. In Kapitel 3 wird die Schreibweise, die Portabilität und die Kompatibilität der Compiler unter sich genau erläutert. Für das Verständnis des Beispiels ist es aber ausreichend, den nachfolgenden Erläuterungen zu folgen.

Damit das Programm Ausgaben tätigen kann, muß es im groben wie folgt vorgehen: Zunächst wird das Arbeitsgerät geöffnet, danach können Ausgaben getätigt werden. Falls Zeichen nicht im Systemzeichensatz ausgegeben werden sollen, müssen einmalig vor der Ausgabe Zeichensätze geladen werden. Nach der Grafikausgabe muß die Arbeitsstation wieder geschlossen werden. Diese Reihenfolge ist für alle Ausgabegeräte gleich.

Am Anfang des Programmtextes werden einige Konstanten für die Ausgabegeräte definiert. Diese Konstanten entsprechen den Gerätekennungen aus der Datei ASSIGN.SYS. Für den Bildschirm gibt es die Kennung 1, für den Plotter 11, für den Drucker 21 und für den Metadatei-Treiber die Kennung 31.

Die beiden Koordinatensysteme NDC und RC werden beim Öffnen der Arbeitsstation verlangt. Die Zeichenkette soll im Zeichensatz Swiss ausgegeben werden, der unter der Nummer 2 zu erreichen ist.

Da das GEM auf dem ATARI die beiden Funktionen vm_pagesize (Metadatei-Seitengröße) und vm_coordinates (Koordinatensystem der Metadatei) nicht direkt kennt, werden diese beiden Funktionen später durch direkten Aufruf des VDI nachgebildet. Sie benutzen die Konstanten VM_PAGESIZE bzw. VM_COORDS.

Bei den Variablen werden zunächst die GEM-Arrays contrl, intin, ptsin, intout und ptsout global erklärt. Sie werden bei VDI-Befehlen durch die sogenannten Bindings oder GEM-Libraries gefüllt.

Die lokalen Variablen work_in und work_out sowie das vdi_handle werden beim Öffnen der Arbeitsstation sowie für VDI-Aufrufe ständig benutzt.

Die Variable from_desktop wird bei einem ATARI ST immer auf TRUE initialisiert, während sie unter MS-DOS mit FALSE vorbelegt werden kann, falls das Beispielprogramm direkt über das VDI aufgerufen werden soll (GEMVDI -VDITEST.EXE, siehe 2.3.5). Wird das Beispiel vom DESKTOP aus gestartet, so ist auch hier der Wert TRUE einzutragen.

Die Variable meta_name beinhaltet den Dateinamen für die Ausgabe auf eine Metadatei, während min_x, min_y, max_x und max_y dazu benutzt werden, um die maximale Ausdehnung unserer Grafik zu bestimmen. Die Variablen screen_w, screen_h sowie meta_w, meta_h werden dazu benutzt, das Koordinatensystem der Metadatei festzulegen, wenn es im Rastersystem geöffnet wird (Siehe auch 2.7.1).

Die Funktion vdi, die als EXTERN erklärt wird, wird nur bei direktem Aufruf des VDI benötigt. Die Prototypen der vier lokalen Funktionen open_work, close_work, wait und house sind nur für ANSI-Compiler wie z.B. Turbo-C wichtig.

Wenn GEMDOS gesetzt ist, also das Beispielprogramm auf einem ATARI ST übersetzt wurde, wird das GEM noch um zwei Funktionen des VDI erweitert. Diese Funktionen sind ab GEM 2.X unter MS-DOS direkt verfügbar.

Die Funktion vm_pagesize setzt in den Kopf einer Metadatei die Papierbreite und Höhe. Der Kopf einer Metadatei enthält wichtige Informationen über die Grafikausdehnung, Papiergröße sowie das benutzte Koordinatensystem. Im Kapitel 2.7.1 wird der Aufbau genau beschrieben.

Die Funktion vm_coords setzt in den Kopf einer Metadatei Informationen über das verwendete Koordinatensystem. Zusammen mit den Informationen über die Papiergröße läßt sich später erst berechnen, wie groß ein Pixel in einer Metadatei ist (siehe Kapitel 2.7.1).

Die Funktion open_work benötigt zwei Parameter. Zum einen möchte sie wissen, welches Arbeitsgerät zu öffnen ist, zum anderen soll das Arbeitsgerät in einem bestimmten Koordinatensystem (NDC, RC) benutzt werden.

Zunächst wird das work_in-Feld mit 0 vorbelegt. Das nullte Element muß mit der Gerätekennung (1, 11, 21 oder 31, siehe ASSIGN.SYS) vorbelegt werden. Das work_in-Array wurde gleich auf die Länge von 103 Elementen gebracht, da ab GEM/3 weitere Angaben gemacht werden können. In work_in [10] wird das gewählte Koordinatensystem eingetragen.

Arbeitet man unter GEM/3, so kann noch angegeben werden, ob das Ausgabegerät auf Datei, serielle oder parallele Schnittstelle oder auf die Standardschnittstelle ausgeben soll. Die Standardschnittstelle wird beim Initialisieren des GEM/3 auf dem PC abgefragt. In work_in [11] geben wir die Zahl 255 an (OW_NOCHANGE aus VDI.H), um die Standardschnittstelle zu wählen.

Ist das Ausgabegerät der Bildschirm, so wird unterschieden, ob man das Programm vom Desktop aus gestartet hat oder direkt über das VDI. Im ersten Fall wird die Applikation initialisiert (wie bei jedem GEM-AES-Programm) und die physikalische Kennung (handle) über graf_handle geholt. Da beim Start des Desktop das GEM schon initialisiert ist und den Bildschirm als physikalisches Ausgabegerät geöffnet hat, müssen wir den Bildschirm virtuell öffnen (v_opnvwk). Jedes Ausgabegerät läßt sich nämlich nur einmal physikalisch öffnen. Anschließend wird die Maus über eine AES-Funktion versteckt.

Im anderen Fall wird der Bildschirm (zum ersten Mal) physikalisch geöffnet und die Maus über die VDI-Funktion v_hide_c versteckt. Die Größe des Bildschirms merken wir uns für die spätere Ausgabe auf eine Metadatei.

Wird nicht der Bildschirm als Ausgabegerät gewählt, so können wir in jedem Fall unser Ausgabegerät physikalisch öffnen (v_opnwk). Wurde das Ausgabegerät erfolgreich geöffnet (vdi_handle größer als 0), so werden noch einige Initialisierungen vorgenommen. Der Bildschirmn wird gelöscht, Plotter und Drucker werden initialisiert. Schreiben wir auf eine Metadatei, so wird der Dateiname, der standardmäßig auf GEMFILE.GEM eingestellt ist, auf die Variable meta_name gesetzt (HOUSENDC.GEM, HOUSERC.GEM).

Da Text im proportionalen Zeichensatz Swiss ausgegeben werden soll, werden die Zeichensätze über vst_load_fonts geladen. Normalerweise müßte man das Funktionsergebnis überprüfen, das aussagt, wieviele Zeichensätze außer dem Systemzeichensatz zur Verfügung stehen. Werden keine Zeichensätze geladen, so erscheint z.B. auf dem Drucker auch nur das Haus, da er über keinen Systemzeichensatz verfügt.

Das erfolgreiche Öffnen der Arbeitsstation (vdi_handle > 0) wird als Funktionswert zurückgegeben und wird in der Funktion house weiterverwendet.

Die Funktion close_work schließt das geöffnet Ausgabegerät wieder. War es der Bildschirm, so wird auf einen Tastendruck gewartet. Mit den entsprechenden AES- oder VDI-Funktionen wird die Maus wieder gezeigt. Wurde vom Desktop gestartet und war das Ausgabegerät der Bildschirm, so werden die geladenen Zeichensätze wieder freigegeben (vst_unload_fonts), das virtuelle Arbeitsgerät geschlossen (v_clsvwk) sowie das AES verlassen (appl_exit).

In allen anderen Fällen werden entsprechend dem Ausgabegerät andere Abschlußhandlungen vorgenommen. Bei Plotter und Drucker wird die Grafik, die wir ausgegeben haben (Haus und Text) und die zunächst vom VDI gepuffert wurde, jetzt aus diesem Puffer auf das Gerät gebracht (v_updwk).

Bei einer Metadatei als Ausgabegerät wird entsprechend dem gewählten Koordinatensystem (NDC/RC) die linke untere Ecke (lower left x/y) sowie die rechte obere Ecke der maximalen Koordinaten innerhalb der Metadatei (upper right x/y) auf folgende Größe gesetzt: Beim NDC-System auf 32766, 32766, beim RC-System auf die aktuelle Bildschirmgröße (z.B. 640, 400). Eigentlich müßten die Werte beim NDC-System 32767 lauten. Leider hat das GEM-Output einen Fehler (bis zur GEM-Version 3.11). Es berechnet die Breite und Höhe mit der üblichen Formel

Breite der Grafik = X2 − X1 + 1 = 32767 − 0 + 1 = 32768
Höhe der Grafik = Y2 − Y1 + 1 = 32767 − 0 + 1 = 32768

Da diese Zahlen aber negativ sind, dreht GEM-Output durch und zeigt gar nichts mehr an. Lediglich GEMDRAW (und das mitgelieferte SHOWGEM) kommen mit diesen Werten zurecht. Verringert man die Ausdehnung um 1 Pixel, so ergibt sich für die Breite und Höhe der Wert 32767, was zwar nicht ganz korrekt, aber einen vernachlässigbaren Fehler ergibt.

Die drei folgenden Metadatei-Funktionen bestimmen Pixelgröße, Papiergröße sowie die Ausdehnungen des Bildes in der Metadatei. Mittels v_meta_extents gibt man die größte Ausdehnung des Bildes an (min_x/y, max_x/y). Dieser Wert wird in der Funktion house gesetzt. Ein Programm, das Metadateien einliest, kann so leicht die maximalen Koordinatenwerte der Grafik feststellen.

Die Funktion vm_pagesize schreibt in die Metadatei, wie groß die Seite in 1/10 mm war, für die die Grafik erstellt wurde. Eine normale Druckerseite im Hochformat z.B. hat $8.5 \times 11$ Zoll, wenn das Bild in GEMDRAW erstellt wurde. Da GEMDRAW einen Rand von 0.5 Zoll um das gesamte Bild freiläßt, ergibt sich eine Größe von $7.5 \times 10$ Zoll. Multipliziert man diese Werte mit 2.54, so ergibt sich eine Größe von $19.05 \times 25.4$ cm bzw. $1905 \times 2540$ in 1/10 mm.

Die Funktion vm_coords schließlich gibt an, wie groß die Auflösung in x- und y-Richtung sein soll. Zusammen mit der Seitengröße (vm_pagesize) kann erst errechnet werden, wieviele Punkte/cm in einer Metadatei dargestellt wurden. Für maßstabsgetreue Abbildungen ist diese Angabe in einer Metadatei unerläßlich. Näheres über Metadateien erfahren Sie in Kapitel 2.7.1. Eine Tabelle aller Standardgrößen und Koordinatenwerte ist dort ebenfalls aufgelistet. Die Bezeichnung vm_pagesize sowie vm_coords sind die offiziellen VDI-Namen und sollten in eigenen Programmen beibehalten werden.

Am Ende werden die geladenen Zeichensätze wieder freigegeben (vst_unload_fonts) und das Arbeitsgerät geschlossen (v_clswk).

Die Funktion wait wartet auf einen Tastendruck, falls das Programm vom Desktop aus gestartet wurde (evnt_keybd). Wird nur das VDI benutzt, so wird über die Funktion vrq_string auf eine Taste gewartet.

Die Funktion house öffnet das gewählte Ausgabegerät (open_work). Im erfolgreichen Falle werden für das NDC/RC-System die Koordinaten für das Haus in das Feld pxy gesetzt. Man beachte, daß im NDC-System der (0,0)-Wert in der linken unteren Ecke liegt, im RC-System in der linken oberen Ecke. Wurde eine Ausgabe auf eine Metadatei gewünscht, so werden hier die Variablen min_x usw. bzw. meta_w und meta_h gesetzt, die beim Schließen des Gerätes benutzt werden. Beim RC-System werden die maximalen Bildschirm-Werte auch für die Metadatei benutzt. Da das OUTPUT einen Fehler beim Berechnen der Größe einer Seite macht, wird die Breite bzw. Höhe im NDC-System „nur" auf 32766 gesetzt. Dies ist eine winzige Ungenauigkeit und kann vernachlässigt werden.

Wurden alle Koordinatenwerte gesetzt, dann wird das Haus mittels der VDI-Funktion v_pline ausgegeben (Strecken- bzw. Polygonzüge). Vorher wird die Linienfarbe auf rot (RED) sowie der Linientyp auf durchgezogen (SOLID) eingestellt. Um den Text auszugeben, müssen verschiedene Voreinstellungen gemacht werden. Der Schreibmodus wird auf transparent gesetzt (vswr_mode), die Textfarbe auf grün (vst_color), der Zeichensatz auf SWISS (vst_font) und die Text-Effekte auf normal gesetzt (vst_effects). Die Textausrichtung soll linksbündig (ALI_LEFT) sein und der Text soll auf der Grundlinie

(ALI_BOTTOM) des Hauses ausgegeben werden. Wir benutzen einen Text der Größe 10-Punkt (vst_point), wobei der Text um 0 Grad rotiert, also von links nach rechts ausgegeben werden soll (vst_rotation). Schließlich erfolgt die eigentliche Ausgabe des Textes „VDI-Test HAUS" im linken unteren Eckpunkt des Hauses (v_gtext). Am Ende wird das Ausgabegerät wieder geschlossen (close_work).

Das Hauptprogramm main setzt die Bildschirmbreite und Höhe auf einen Default-Wert. Er wird aber korrigiert, falls das Haus auf dem Bildschirm ausgegeben werden soll. Anschließend wird die Variable from_desktop vorbelegt. Startet man das Programm vom Desktop aus (ATARI ST oder IBM-PC), so muß der Wert TRUE sein. Startet man das Programm unter MS-DOS direkt über das VDI (GEMVDI /VDITEST.EXE bei GEM 2.X bzw. GEMVDI -VDITEST.EXE bei GEM/3) so muß dieser Wert auf FALSE initialisiert werden.

Jetzt wird das Haus auf den verschiedenen Geräten mit den beiden Koordinatensystemen ausgegeben. Die Reihenfolge ist Bildschirm, Plotter, Drucker und Metadatei. Soll die Ausgabe nur auf Bildschirm und Metadatei erfolgen, weil kein Plotter oder Drucker angeschlossen ist, so kann man nach Belieben die Funktionsaufrufe auskommentieren.

Zusammenfassung:

Das Programm VDITEST.C zeigt anschaulich, daß die Grafikausgabe auf verschiedene Geräte im Prinzip immer gleich erfolgt. Zuerst wird das entsprechende Gerät geöffnet, wobei man die VDI-Kennung (vdi_handle) bekommt. Mittels Angabe dieser Kennung können Grafikausgaben getätigt werden. Am Ende wird das Ausgabegerät wieder geschlossen.

Für verschiedene Ausgabegeräte muß man allerdings einige Sonderfälle berücksichtigen. Benutzt man den Desktop zum Starten von VDI-Programmen, so ist der Bildschirm vom GEM-System schon vorher physikalisch geöffnet (v_opnwk) worden. In diesem Fall kann der Bildschirm nur noch virtuell geöffnet werden (v_opnvwk). Dies ist für den Programmierer kein Nachteil, da sich nach dem virtuellen Öffnen das VDI genauso verhält, als ob man den Bildschirm physikalisch geöffnet hätte.

Plotter, Drucker, Kameras etc. werden alle gleich behandelt. Eine Ausnahme bilden nur die Metadateien. Hier kann man nach dem Öffnen des Ausgabegerätes noch angeben, wie die Metadatei heißen soll (vm_filename). Bei der Grafikausgabe sollte man sich dann noch einige Werte merken:

a) die Ausdehnung der Grafik
b) die Größe der Seite, auf der Ausgaben getätigt wurden
c) die maximalen x-y-Werte des Koordinatensystems

Vor dem Schließen des Ausgabegerätes Metafile sollten in jedem Falle die Werte unter a) – c) eingetragen werden. Dafür existieren die drei VDI-Funktionen für Metadateien

v_meta_extents, vm_pagesize, vm_coords. Wird dies vergessen, so kann das Programm, welches die produzierte Grafik übernehmen möchte, keine Aussagen mehr darüber machen, wie groß ein Pixel aus der Metadatei ist. Maßstabsgetreue Abbildungen sind dann nicht mehr möglich!

Zwei Anmerkungen zum Schluß: Auf dem ATARI ST muß bei Verwendung von Turbo-C das Assemblerprogramm VDICALL.S dazugelinkt werden (VDITEST.PRJ), da der direkte Aufruf des VDI anders implementiert wurde als bei allen anderen Compilern. Schade drum. Borland hätte sich erst bei anderen Compilerbauern informieren sollen.

Das GDOS auf dem ATARI ST hat außerdem einen Fehler: Öffnet man Metadateien im NDC-Koordinatensystem, so werden alle y-Werte von 32767 subtrahiert. Dies ist ein grober Fehler. Grafikdateien sind dann nämlich unbrauchbar (das Haus steht auf dem Kopf!). Es empfiehlt sich also, Metadateien immer im RC-System zu öffnen, zumindest so lange, bis ein funktionierendes GDOS auf dem ST existiert. Das Benutzen des RC-Systems stellt aber keinen Nachteil dar, da auch hier x- und y-Koordinatenwerte bis 32767 möglich sind. Lediglich der Ursprung (0,0) liegt in der linken oberen Ecke, was bei den meisten bekannten Grafikpaketen benutzt wird. Die beiden korrekt erzeugten Metadateien HOUSENDC.GEM und HOUSERC.GEM werden auf Diskette mitgeliefert. Sie wurden unter MS-DOS mit GEM/3 erstellt.

### 2.3.7 Funktionen der VDI-Gruppen

Die VDI-Funktionen wurden schon in sehr vielen Büchern beschrieben. Aber der erklärende Text jeder einzelnen Funktion war meist nur 1 oder 2 Zeilen lang. Oft wurde vieles auch falsch erklärt. Es sollen an dieser Stelle alle Funktionen, die eine ausführliche Erklärung benötigen, genau besprochen werden. Funktionen, die keine große Bedeutung haben, wie z.B. das Zeichnen eines Kreises, werden nur der Vollständigkeit halber aufgelistet.

Zum Nachschlagen der Parameter jeder einzelnen Funktion kann ein x-beliebiges GEM-Buch benutzt werden. Wir empfehlen das Buch aus dem gleichen Verlag (Softwareentwicklung auf dem ATARI ST), das alle GEM-Funktionen bis GEM 2.X beschreibt. Parameter werden nur dort erläutert, wo es für das Verständnis wichtig erscheint. Außerdem werden alle neuen GEM/3-Funktionen mit Parametern genau erläutert, da diese in keinem einzigen Buch für den ATARI ST erwähnt werden, aber für den einen oder anderen interessant sind, der Programme für ATARI und MS-DOS erstellen möchten.

Zu jeder Gruppe werden in einer Tabelle kurz alle Funktionen mit ihrem Zweck aufgelistet. Vor dem C-Funktionsnamen werden die Funktionsnummern angegeben.

Gruppe 1: Kontrollfunktionen

Sie werden benötigt, um Arbeitsgeräte zu initialisieren und Zeichensätze für ein Gerät zu laden.

| | | |
|---|---|---|
| 1 | v_opnwk | Öffnen einer Arbeitsstation |
| 2 | v_clswk | Schließen einer Arbeitsstation |
| 3 | v_clrwk | Löschen einer Arbeitsstation |
| 4 | v_updwk | gepufferte Grafikkommandos ausgeben |
| 100 | v_opnvwk | virtuelle Arbeitsstation öffnen |
| 101 | v_clsvwk | virtuelle Arbeitsstation schließen |
| 119 | vst_load_fonts | Zeichensätze nachladen |
| 119 | vst_ex_load_fonts | Zeichensätze nachladen (GEM/3) |
| 120 | vst_unload_fonts | Zeichensätze freigeben |
| 129 | vs_clip | Clipping aus-/einschalten |

– V_OPNWK

Dies ist eine der zentralen Funktionen. Man übergibt die Gerätenummer (siehe 2.3.4) und kann einige Grundeinstellungen für das Ausgabegerät vornehmen (Linienfarbe, Textfarbe, Linientyp etc). Beim Aufrufen der Funktion wird in der Datei ASSIGN.SYS nachgeschaut (bzw. in der Zuweisungstabelle), ob ein Treiber für das Gerät verfügbar ist (z.B. Plotter). Ist er vorhanden, so wird er geladen und mit den Einstellungen aus dem work_in-Feld initialisiert.

In diesem Falle bekommt man eine Kennung (vdi_handle) zurück, mit der man alle anderen VDI-Funktionen aufrufen kann. Das System erkennt anhand dieser Kennung das angesprochene Gerät. Zu jeder Kennung existiert intern ein Satz von Attributen, die zum einen durch das work_in-Feld, zum anderen automatisch initialisiert wurden. Diese Initialisierungen, die nicht aus dem work_in-Array kommen, sind wie folgt definiert:

| | |
|---|---|
| Textgrundlinie | 0 Grad |
| Textausrichtung | links |
| Texteffekt | 0 |
| Strichdicke | 1 |
| Polyline-Enden | quadratisch |
| Schreibmodus | ersetzen (replace) |
| Eingabe-Modus | request |
| Umrandung (perimeter) | eingeschaltet |
| Benutzerdefinierter Linientyp | durchgezogen |
| Benutzerdefiniertes Füllmuster | DRI-/ATARI-Logo |
| Cursor | versteckt |
| Clipping | ausgeschaltet |

Es gibt 16 feste Farben in GEM, die über die ersten 16 Indizes angesprochen werden. Diese 16 Farben sollten in jedem Falle auf jedem Rechner gleich eingestellt werden, da dann Farbgrafiken unter verschiedenen Rechnern ohne Probleme ausgetauscht werden können. Farben ab Index 16 aufwärts können frei benutzt werden. Die ersten 16 Farben werden von Digital Research wie folgt definiert und gesetzt:

| Index | Farbe |
|---|---|
| 0 | weiß (Hintergrundfarbe) |
| 1 | schwarz |
| 2 | rot |
| 3 | grün |
| 4 | blau |
| 5 | cyan (türkis) |
| 6 | gelb |
| 7 | magenta (violett) |
| 8 | hellgrau |
| 9 | dunkelgrau |
| 10 | dunkelrot |
| 11 | dunkelgrün |
| 12 | dunkelblau |
| 13 | dunkelcyan |
| 14 | dunkelgelb |
| 15 | dunkelmagenta |
| 16 – n | geräteabhängig |

Wie diese Einstellungen auf einem ATARI ST in niedriger Auflösung erreicht werden, wird bei der Funktion Copy-Raster-Form erläutert.

Nach dem Aufruf der Funktion v_opnwk befinden sich im work_out-Feld äußerst wichtige Werte. Sie sind entscheidend für das Schreiben von Programmen, die in einer unabhängigen Umgebung (Bildschirmauflösung, Farben etc.) laufen sollen.

Hier gleich ein Hinweis für alle ST-Programmierer: Das Programm BIGSCREEN von Julian Reschke (Markt & Technik) simuliert einen beliebig großen Bildschirm auf dem ST. Das Programm läuft auch in Farbe. So kann man sich z.B. eine Auflösung von 640x400 Punkten mit 16 Farben herbeizaubern. Beim Testen stießen wir leider nur auf eine handvoll Programme, die mit dieser Auflösung arbeiten (z.B. alle DRI-Programme wie GEMDRAW, RCS, ADIMENS ST und WORDPLUS). Es gibt etliche Programme, die nur in der mittleren Auflösung laufen, da sie meinen, sie hätten dann eine Menüleiste mit 80 Zeichen. Das ist aber falsch und liegt daran, daß diese Programme die XBIOS-Funktion Getrez() benutzen, um die Auflösung festzustellen. Wir appellieren daher an alle ST-Programmierer, Getrez() nur noch dann zu benutzen, wenn das Programm wirklich die Auflösung von 320x200 Punkten benötigt oder direkt in das Video-RAM schreibt. Alle anderen Fälle können mit einer Abfrage der work_out [0]- und work_out [1]-Werte auskommen.

Ist das Ausgabegerät beim Öffnen der Arbeitsstation eine Metadatei, so wird der Kopf der Meta-Datei (18 VDI-Grundeinstellungen) geschrieben. Diese 18 Grundeinstellungen können mit dem Programm DUMPMETA.EXE bzw. DUMPMETA.TOS im Klartext ausgegeben werden. Jeder weitere VDI-Befehl mit der zurückgelieferten Kennung wird dann in die Meta-Datei geschrieben.

Ist die Kennung 0, so konnte das Gerät nicht geöffnet werden. In allen anderen Fällen bekommt man im work_out-Feld 57 Werte zurück, von denen einige hier erläutert werden sollen:

work_out [0], work_out [1] (s.o.: Bemerkung für ST-Programmierer)
work_out [2]: 0 = Gerät kann präzise skalieren
1 = Gerät kann nicht präzise skalieren
work_out [3], workout [4]: Pixelbreite und -höhe in 1/1000 mm

Aus den Werten work_out [2] bis work_out [4] kann man bestimmen, ob man maßstabsgerechte Zeichnungen ausgeben kann. Für einen Drucker z.B. ist der work_out [2]-Wert 0. Man kann also mit Hilfe der Pixelbreite- und -höhe genau sagen, wieviele Pixel man ausgeben muß, um z.B. eine Linie von 1 cm Länge zu zeichnen.

Bei Bildschirmen ist der work_out [2]-Wert 1. Das bedeutet, daß man nicht genau sagen kann, wieviele Pixel man setzen muß, um eine Linie von 1 cm auf einem Bildschirm zu zeichnen. Zwar wird die Pixelbreite und -höhe zurückgegeben, aber es handelt sich hier nur um einen ungefähren Wert, da der Hersteller eines Bildschirmtreibers ja nicht wissen kann, welcher Monitor am Rechner angeschlossen wird. Die Bildschirmdiagonalen sind ja von Monitor zu Monitor verschieden. Das bedeutet, daß man aus den Pixelbreiten und -höhen lediglich ablesen kann, ob eine Verzerrung des Bildes stattfindet. Ist die Pixelbreite z.B. 400/1000 mm und die Pixelhöhe 200/1000 mm, so weiß man, daß die y-Koordinaten eines Quadrates immer halb so groß sein müssen wie die x-Koordinaten. Man kann also immerhin das Seitenverhältnis (aspect ratio) bestimmen.

Wer die genauen Werte haben möchte, kann ja seinen Monitor ausmessen und die Breite sowie Höhe des Monitorbildes durch die Pixelanzahl teilen. Mit diesen Werten kann man maßstabsgetreu auf dem Bildschirm zeichnen.

GEM/3 Hinweis:

Die Funktion v_opnwk wurde geändert (siehe auch VDITEST.C). Das work_in-Feld wurde erweitert, so daß zur Laufzeit noch einige Einstellungen für den Treiber vorgenommen werden können. Falls jemand für den ATARI ST einen Treiber schreiben möchte, so sollte er diese Informationen verwenden, um kompatibel mit zukünftigen GEM-Versionen zu sein.

work_in [11]: niederwertiges Byte = Ausgabe auf

```
  0 = Datei
  1 = seriellen Port
  2 = parallelen Port
  3 = gerätespezifisch (direkt)
255 = Default-Schnittstelle
```

höherwertiges Byte = Seitengröße

```
  0 = Letter size  ( 8.50 x 11.00 Zoll)
  5 = Half size    ( 8.50 x  5.50 Zoll)
 10 = B5 size      (18.20 x 25.00 cm)
 20 = Letter size  ( 8.50 x 11.00 Zoll)
 30 = A4 size      (21.00 x 29.70 cm)
 40 = Legal size   ( 8.50 x 14.00 Zoll)
 50 = Double size  (11.00 x 17.00 Zoll)
 55 = Broad sheet  (14.00 x 11.00 Zoll)
255 = Userdef      (work_in [101..102])
```

Konstanten für die obigen Werte befinden sich im VDI.H und sollten benutzt werden.

work_in [12] – work_in [100]: Ausgabeport bzw. Dateiname

Ist work_in [11] = 1 oder 2, so muß work_in [12] auf die Port-Nummer gestellt werden (LPT1 = 0, LPT2 = 1, COM1 = 0, etc). Ist work_in [11] = 0, so muß sich von work_in [12] an der Dateiname befinden, auf den geschrieben werden soll. Der Dateiname wird Zeichen für Zeichen in die 16-Bit-Worte work_in [12-....] geschrieben und muß mit einer 0 enden.

C-Beispiel:

```
WORD work_in [103]; /* Man beachte die neue Länge */
BYTE filename [80];
WORD i;

work_in [11] = 0;
strcpy (filename, "D:\\TMP\\PRINT");
for (i = 0; i <= strlen (filename); i++)
  work_in [i + 12] = filename [i];
```

Der Vorteil liegt auf der Hand: Man stelle sich vor, es gäbe ein Accessory, das man System-Spooler nennt. Eine Ausgabe von einem beliebigen Programm könnte statt direkt auf den Drucker zunächst auf eine Datei gehen. Anschließend wird dem Accessory der Dateiname mitgeteilt (appl_write). Dieses druckt im Hintergrund Byte für Byte die genannte Datei, da sie ja schon im Ausgabeformat vorliegt. Man könnte dann ohne Probleme weiterarbeiten. In der Tat wird dies z.B. in der GEM/3-Version gemacht. Dort kann

man dem OUTPUT-Programm sagen, daß es im Hintergrund drucken soll. Es schreibt die Ausgabe auf eine temporäre Datei und schickt dem Accessory CALCLOCK.ACC eine entsprechende Meldung. Im AES wird noch einmal darauf eingegangen. Dort wird gezeigt, wie die Meldung für den System-Spooler aussehen muß. Falls ein Leser gerne einen System-Spooler für den ATARI ST entwickeln möchte, so sollte er sich an die Konventionen in Kapitel 2.4 halten.

work_in [101]: Seitenbreite in 1/100 Zoll
work_in [102]: Seitenhöhe in 1/100 Zoll

Im work_out-Feld werden folgende zusätzlichen Informationen ausgegeben:

work_out [14]: Ist 11, wenn „Escapement text" verfügbar ist
work_out [24]: Ist 11, wenn „Escapement text" der einzig verfügbare Text auf dem Ausgabegerät ist. Sonst wird 10 zurückgegeben. Escapement text wird weiter unten erklärt.

work_out [44]: -1 bedeutet, daß das Ausgabegerät im Querformat ausgeben kann, ohne daß man selbst die Grafik drehen muß. Das OUTPUT-Programm übernimmt beim Querformat das Drehen allerdings selbst, da es diese Möglichkeit in älteren GEM-Versionen nicht gab.

– V_CLSWK

Die angegebene Arbeitsstation wird geschlossen. Bei Bildschirmen wird auf Textdarstellung umgeschaltet, Plotter, Drucker und Kamera senden ihre gepufferten Grafikbefehle auf das physikalische Gerät, bei Metadateien wird ein EOF-Zeichen (End-Of-File) mit dem Wert $FFFF geschrieben und die Datei geschlossen. Vor Benutzen dieses Befehls müssen alle eventuell virtuell geöffneten Arbeitsgeräte auch wieder virtuell geschlossen werden.

– V_CLRWK

Die Arbeitsstation wird gelöscht. Bei Bildschirm und Kamera wird mit der Hintergrundfarbe gelöscht, beim Plotter wird der Puffer gelöscht, bei einem Drucker wird der Puffer gelöscht und ein Seitenvorschub ausgegeben. Bei einer Metadatei wird der Befehl in die Datei geschrieben.

– V_UPDWK

Alle gepufferten Ausgaben werden jetzt an das Gerät geschickt. Man muß vor der Druckerausgabe einmalig v_updwk aufrufen, um den Drucker zu initialisieren.

- V_OPNVWK
- V_CLSVWK

Bildschirme können von mehreren Programmen (z.B. eine Applikation, mehrere Accessories) gleichzeitig benutzt werden. Der Bildschirm darf allerdings im System nur einmal physikalisch geöffnet werden (v_opnwk). Dies wird beim Initialisieren des GEM getan, wenn man mit dem AES bzw. dem Desktop arbeitet. Es gibt aber die Möglichkeit, den Bildschirm öfters zu benutzen als nur von einer Applikation aus. Dazu kann er beliebig oft virtuell geöffnet werden. Im Prinzip ist dies die gleiche Funktion wie das physikalische Öffnen, mit dem Unterschied, daß man die Kennung, die beim physikalischen Öffnen zurückgegeben wurde, jetzt als Eingabeparameter mit übergibt.

Beim virtuellen Öffnen des Bildschirms legt der Bildschirmtreiber jedesmal einen neuen Satz von Attributen (z.B. Liniendicke- und -farbe, Zeichensatzhöhe, Schreibmodi usw.) an. Verschiedene Programme können über die Kennung, die zurückgegeben wird, dann diese Attribute ändern.

Zeichnet z.B. ein Programm eine blaue Linie, so setzt es die Linienfarbe auf Blau (vsl_color) und tätigt die Ausgabe. Ein Accessory kann zwischenzeitlich eine rote Linie zeichnen. Es stellt die Linienfarbe also auf Rot. Zeichnet die Applikation nun wieder eine Linie, so wird ihre Farbe blau sein, obwohl zwischenzeitlich auf rot gestellt wurde. Der Bildschirmtreiber kann anhand der Kennung (vdi_handle) also feststellen, welche Linienfarbe zu welcher Applikation gehört.

Entsprechend werden virtuelle Ausgabegeräte mit v_clsvwk wieder geschlossen.

- VST_LOAD_FONTS
- VST_EX_LOAD_FONTS (GEM/3)
- VST_UNLOAD_FONTS

Mit diesen Befehlen ist es möglich, zusätzliche Zeichensätze für ein Arbeitsgerät zu laden. Wie schon weiter oben erklärt, werden die Namen der Sätze aus der Datei ASSIGN.SYS bzw. aus dem \GEMAPPS\FONTS-Ordner bei GEM/3 geholt.

Ein C-Aufruf mit

```
WORD additional;

additional = vst_load_fonts (vdi_handle, 0)
```

lädt alle Zeichensätze, die zusätzlich außer dem System-Zeichensatz zur Verfügung stehen. Der Wert additional gibt an, wieviele zusätzliche Sätze geladen werden konnten.

Ab GEM/3 ist es möglich, Zeichensätze dynamisch ein- und auszulagern (extended load). Das GEM übernimmt hier das Font-Swapping. Man übergibt beim Laden der Sätze,

wieviel Speicher man maximal mit ihnen belegen möchte und wieviel Speicher man minimal frei haben möchte.

Der Aufruf hat dann zwei Parameter mehr:

additional = vst_ex_load_fonts (vdi_handle, select, font_max, font_free);

Select muß wie immer 0 sein, font_max gibt den maximal zu reservierenden Speicher an und font_free den minimalen Speicher, der frei gelassen werden soll.

Default-Werte sind:

font_max:
Bildschirm: 5120 ( 80 KByte)
Drucker : 32767 (512 KByte)

font_free :
Bildschirm: 0 ( 0 KByte)
Drucker : 640 (10 KByte)

Mit vst_unload_fonts werden die zusätzlich geladenen Zeichensätze wieder aus dem Speicher entfernt.

– VS_CLIP

Da mehrere Applikationen den Bildschirm als Ausgabegerät benutzen können, muß es eine Möglichkeit geben, die Grafikausgaben so einzuschränken, daß diese nur in einem bestimmten Bereich des Bildschirms wirken. Meist werden bei GEM-Programmen sogenannte Bildschirmfenster benutzt. Der Bereich, auf den man ausgeben möchte, wäre dann das Innere dieses Fensters.

Zu diesem Zweck kann ein sogenanntes Clipping-Rechteck gesetzt werden. Alle Ausgaben werden dann nur innerhalb dieses Rechteckes ausgeführt. Ist das Rechteck z.B. 100x100 Pixel groß und wird eine horizontale Line mit Länge 200 Pixel mitten durch das Rechteck gezogen, so wird die Linie außen abgeschnitten, und nur der im Rechteck liegende Teil wird gezeichnet. Das Clipping läßt sich auch wieder ausschalten.

Gruppe 2: Ausgabefunktionen

Ausgabefunktionen für grafische Elemente wie Linien, Kreise, Ellipsen etc. stehen hier zur Verfügung. Beim Aufruf der Funktionen werden noch die zugehörigen Attribute benutzt. Z.B. kann die Linienfarbe auf Rot gestellt werden, wobei dann alle nachfolgenden Zeichenfunktionen, die Linien betreffen, mit der Farbe Rot ausgegeben werden.

| | | |
|---|---|---|
| 6 | v_pline | Linienzug |
| 7 | v_pmarker | Markierungspunkt setzen |
| 8 | v_gtext | Text ausgeben |
| 9 | v_fillarea | Linienzug mit Fläche füllen |
| 10 | v_cellarray | Raster aus Zellen aufbauen |
| 11, 1 | v_bar | Rechteck |
| 11, 2 | v_arc | Kreisauschnitt |
| 11, 3 | v_pieslice | gefüllter Kreisauschnitt |
| 11, 4 | v_circle | gefüllter Kreis |
| 11, 5 | v_ellipse | gefüllte Ellipse |
| 11, 6 | v_ellarc | elliptischer Kreisbogen |
| 11, 7 | v_ellpie | gefüllter elliptischer Kreisbogen |
| 11, 8 | v_rbox | Rechteck mit abgerundeten Ecken |
| 11, 9 | v_rfbox | gefülltes abgerundetes Rechteck |
| 11,10 | v_justified | ausgerichteter Text |
| 11,11 | v_etext (GEM/3) | erweiterte Textausgabe |
| 103 | v_contourfill | Fläche füllen |
| 114 | vr_recfl | gefülltes Rechteck ohne Rand |

Da bei den verschiedenen Ausgabefunktionen Attribute (Liniendicke, Füllfarbe usw.) eingestellt werden müssen, werden sie nachfolgend tabellenartig aufgelistet. Somit sieht man sofort, welche Attribute auf welche Ausgabefunktionen wirken.

a) Linienattribute

Sie werden bei den Funktionen v_pline, v_arc, v_ellarc sowie v_rbox benutzt.

vsl_type = Linientyp (z.B. gestrichelt)
vsl_width = Liniendicke
vsl_color = Linienfarbe
vsl_ends = Linienenden (Pfeile etc).

b) Attribute für Markierungspunkte (v_pmarker)

vsm_type = Typ einer Marke (z.B. Kreuz)
vsm_height = Größe einer Marke
vsm_color = Farbe einer Marke

c) Attribute für flächenweises Ausgeben

Sie werden bei den Funktionen v_fillarea, v_bar, v_pieslice, v_circle, v_ellipse, v_ellpie, v_rfbox, v_contourfill und vr_recfl benutzt.

vsf_interior = Füllart (z.B. schraffiert)
vsf_style = Füllstil (z.B. Musterart, Schraffurart)
vsf_color = Füllfarbe
vsf_perimeter = Umrandung (nicht bei vr_recfl)

d) Attribute für Textausgaben

vst_height = Schrifthöhe
vst_rotation = Schriftrichtung (z.B. 90 Grad)
vst_font = Zeichensatz
vst_color = Textfarbe
vst_alignment = Textausrichtung (z.B. zentriert)
vst_effects = Texteffekte (z.B. kursiv, fett)
vst_point = Schrifthöhe in Punkten (1 Punkt = 1/72 Zoll)

Für alle Ausgabefunktionen ist die Einstellung des Schreibmodus (vswr_mode) entscheidend. Er bestimmt, wie die Ausgabe durchgeführt wird (transparent, Exklusiv-Oder etc.). Die einzelnen Ausgabefunktionen werden an dieser Stelle nicht mehr besprochen, da der Name der Funktion meist aussagt, um was es sich handelt (v_circle zeichnet natürlich einen Kreis).

Ein kurzes C-Beispiel zeigt, wie die Attributefunktionen mit den Ausgabefunktionen benutzt werden. Es soll ein roter Kreis an der Stelle (100,100) mit Radius 50 und einem schraffierten Muster (hatch) gezeichnet werden, ohne Berücksichtigung des alten Hintergrundes (Replace-Modus). Der Kreis soll eine Umrandung besitzen (perimeter).

```
x = 100;
y = 100;
r = 50;
vswr_mode (vdi_handle, MD_REPLACE);
vsf_interior (vdi_handle, FIS_HATCH);
vsf_style (vdi_handle, 1);
vsf_color (vdi_handle, RED);
vsf_perimeter (vdi_handle, TRUE, SOLID);
v_circle (vdi_handle, x, y, r);
```

– V_ETEXT

Die einzige Funktion, die hier genau erläutert werden soll, ist V_ETEXT, da sie ab GEM/3 zur Verfügung steht. Hiermit ist es möglich, einen Text ab einer gewissen Startposition so auszugeben, daß jedes einzelne Zeichen durch seine Distanz (offset) zu dieser Startposition ausgegeben wird.

Der Aufruf lautet:

```
WORD vdi_handle, x, y;
BYTE string [n];
WORD offsets [n];

v_etext (vdi_handle, x, y, string, offsets);
```

Die x- und y-Koordinate ist der Startpunkt der Textausgabe, string ist eine Zeichenkette, die den Text enthält. In offsets werden soviele Koordinatenelemente angegeben, wie sich Zeichen im string befinden. Der erste Buchstabe benutzt offsets [0] und offsets [1] für die Verschiebung in x- bzw. y-Richtung. Der zweite Buchstabe benutzt offsets [2] und offsets [3] usw. Ein Beispiel soll zeigen, wie man die Proportionalschrift wieder entfernt.

```
strcpy (string, "Nicht proportional");
for (i = 0; i < strlen (string); i++)
{
  offsets [2 * i] = i * 16; /* x-Offset */
  offsets [2 * i + 1] = 0;  /* y-Offset */
} /* for */
v_etext (100, 100, string, offsets);
```

Jeder einzelne Buchstabe wird ab Koordinate (100,100) um 16 Pixel weiter nach rechts ausgegeben. Das 'N' erscheint also an Position (100,100), das 'i' an Position (116, 100) usw. Eine Textausgabe wie

```
G
   E
      M
```

wäre dann auch denkbar.

– V_JUSTIFIED

Diese Funktion, die ausgerichteten Text auf eine angegebene Länge ausgibt, hat sich ab GEM/3 erheblich verändert. Sie wird deshalb ebenfalls genauer erläutert. Der Parameter char_width wurde neu aufgenommen. Er gibt die Anzahl der Pixel jedes Buchstabens der ausgegebenen Zeichenkette zurück. Diese Ausgabeinformationen lassen sich auch abschalten. Die C-Syntax lautet:

```
WORD vdi_handle;
WORD x, y, length;
WORD word_space, char_space;
BYTE string [n];
WORD char_width [n];
```

```
v_justified (vdi_handle, x, y, string, length,
             word_space, char_space, char_width);
```

Die beiden Parameter word_space und char_space haben sich wie folgt geändert:

word_space:
0x0000 = Leerzeichen zwischen Wörtern werden nicht verändert, Ausgabeinformationen (char_width) werden nicht zurückgegeben.

0x0001 = Leerzeichen zwischen Wörtern werden verändert, Ausgabeinformationen (char_width) werden nicht zurückgegeben.

0x8000 = Leerzeichen zwischen Wörtern werden nicht verändert, Ausgabeinformationen (char_width) werden zurückgegeben.

0x8001 = Leerzeichen zwischen Wörtern werden verändert, Ausgabeinformationen (char_width) werden zurückgegeben.

char_space:
0x0000 = Leerstellen zwischen Buchstaben werden nicht verändert
0x0001 = Leerstellen zwischen Buchstaben werden verändert

Im Feld char_width stehen bei entsprechender Angabe von word_space die Pixelbreiten pro Zeichen. Das C-Beispiel zeigt, wie man die Informationen bekommt.

```
BYTE *s;
WORD gesamt;

s      = "Test";
gesamt = 0;
v_justified (vdi_handle, 100, 100, s, 50,
             0x8000, FALSE, char_width);
for (i = 0; i < strlen (s); i++) gesamt += char_width [i];
```

Die Variable gesamt enthält jetzt die Gesamtbreite der vier Zeichen der Zeichenkette „TEST".

Gruppe 3: Attributfunktionen

Die Attributfunktionen kontrollieren alle Ausgabefunktionen. Mit ihnen ist es möglich, Farben, Strichdicken und -arten, Füllmuster und Zeichensatzgrößen zu verändern. Der sogenannte Schreibmodus (writing mode) bestimmt, wie zu setzende Pixel auf schon vorhandene Pixel gesetzt werden.

| | | |
|---|---|---|
| 12 | vst_height | Zeichensatzgröße in Pixel |
| 13 | vst_rotation | Richtung für Zeichenausgabe |
| 14 | vs_color | Farbrepräsentation (Farbtabellen) |
| 15 | vsl_type | Linientyp |
| 16 | vsl_width | Liniendicke |
| 17 | vsl_color | Linienfarbe |
| 18 | vsm_type | Typ der Markierungen |
| 19 | vsm_height | Höhe von Markierungen |
| 20 | vsm_color | Farbe von Markierungen |
| 21 | vst_font | Zeichensatz wählen |
| 22 | vst_color | Farbe von Zeichen |
| 23 | vsf_interior | Füllart |
| 24 | vsf_style | Füllstil |
| 25 | vsf_color | Füllfarbe |
| 32 | vswr_mode | Schreibmodus |
| 39 | vst_alignment | Textausrichtung |
| 104 | vsf_perimeter | Rand ein-/ausschalten |
| 106 | vst_effects | Texteffekte (kursiv, fett etc.) |
| 107 | vst_point | Zeichensatzgröße in Punkt (1/72 Zoll) |
| 108 | vsl_ends | Typ von Linienenden |
| 112 | vsf_udpat | Benutzereigenes Füllmuster |
| 113 | vsl_udsty | Benutzereigener Linientyp |

Einige wichtige Funktionen sollen hier erläutert werden. Alle anderen Funktionen sind trivial und werden in anderer Literatur ausreichend erklärt. An dieser Stelle sei nochmals vermerkt, daß dieses Buch eine Ergänzung zu anderen GEM-Büchern darstellt und nur die Funktionen und GEM-Aspekte hier ausführlich erläutert werden, die in anderen Büchern stiefmütterlich behandelt wurden.

### – VST_HEIGHT

Hiermit kann der aktuelle Zeichensatz in der Ausgabegröße verstellt werden. Die Größe, die man angeben muß, ist die Anzahl der Pixel von der Grundlinie des Zeichens (base line) bis zum oberen Rand (top line). Die Pixelangabe bezieht sich immer auf das gewählte Koordinatensystem (NDC/RC).

Ist das Ausgabegerät der Bildschirm, so versucht das VDI die gewünschte Zeichensatzhöhe zu wählen. Liegt kein Zeichensatz in der gewünschten Größe vor, so werden die Zeichen in der Größe über einen Algorithmus angepaßt. Diese Zeichen sehen dann meist nicht mehr besonders ästhetisch aus. Das OUTPUT-Programm geht diesen Weg, wenn es Meta-Dateien auf dem Bildschirm anzeigt, wo die Ausgabequalität auch keine große Rolle spielt.

Auf allen anderen Ausgabegeräten (z.B. Drucker) werden die Zeichensatzgrößen nicht angepaßt. Der Ausgabetreiber wählt dann den nächstkleineren Satz, um sicherzustellen, daß die Zeichen eine gewisse Ausdehnung nicht überschreiten.

– VST_POINT

Diese Funktion ist ähnlich wie vst_height. Der Unterschied liegt zum einen in der Bemaßung der Größe und zum anderen in der Angabe der Größe.

Der Bemaßung liegt der Begriff „Punktgröße" zugrunde. Ein „Punkt" wird als 1/72 Zoll (= 0.35277 mm) interpretiert. Sollen Zeichen z.B. 1/2 Zoll groß sein, so stellt man die Punktgröße auf 36.

Die Angabe der Größe bezieht sich hier nicht auf den Abstand von der Grundlinie zum oberen Rand, sondern von der Grundlinie einer Textzeile zur Grundlinie der nächsten Textzeile. Diese Größe wird auch Höhe einer Zeichenzelle (im Gegensatz zur Höhe eines Zeichens) genannt. Eine Zeichenzelle ist ein Rechteck, in das das gesamte Zeichen einschließlich Ober- und Unterlänge hineinpaßt. Der Algorithmus des VDI zum Anpassen der Größe, wie in vst_height beschrieben, wird hier nicht angewendet.

_ VS_COLOR

Mit Hilfe dieser Funktion ist es möglich, die allgemeine Farbdarstellung zu verändern. Farben unter GEM werden über Indizes angesprochen (siehe V_OPNWK). Index 0 ist die Hintergrundfarbe (meist weiß), Index 1 ist schwarz, Index 2 ist rot usw. Es ist aber z.B. auch möglich, dem Index 2 die Farbe grün zuzuordnen. Dies wird intern durch Benutzen von sogenannten Farbtabellen (lookup tables) bewerkstelligt. Sie enthalten zu jedem Farbindex eine rechnerabhängige Einstellung der drei Werte für die Grundfarben Rot, Grün und Blau. Mit Hilfe dieser Farben ist es möglich, jeden beliebigen Farbton zu mischen.

Die Anzahl der Werte pro Grundfarbe bestimmt, aus wievielen Farben man einen Farbindex zusammenstellen kann. Bei einem ATARI ST z.B. können in der niederen Auflösung pro Grundfarbe 8 Einstellungen vorgenommen werden. Daraus ergeben sich die Anzahl der Farben zu 8x8x8 = 512 (8 verschiedene Rot-, Grün- und Blauwerte). Da in der niederen Auflösung 16 Farben gleichzeitig darstellbar sind, ist es möglich, jedem dieser 16 Farbindizes einen RGB-Wert zuzuordnen.

Die Anzahl der Werte pro Grundfarbe steigt bei immer leistungsfähigerer Hardware an. So gibt es heute schon Rechner (z.B. Apple Macintosh II) mit 256 Stufen pro Grundfarbe, was 256x256x256 = 16777216 oder 16,7 Millionen Farben ergibt, die sich gleichzeitig darstellen lassen.

Eine Grafik mit über 16 Millionen Farben wirkt schon so realistisch, daß man zum Fersehbild keinen Unterschied mehr bemerkt. Mit der Funktion vs_color ist es möglich, zu jedem Farbindex (z.B. 0 – 15) die RGB-Werte einzustellen. Um auch in Zukunft gewappnet zu sein, entschied man sich, jeden Wert auf ein Promille genau setzen zu können. 0 bedeutet also geringste Farbintensität, 1000 bedeutet höchste Farbintensität. Ein reines Grün hat dann die RGB-Werte (0,1000,0). Damit kann zu jedem Farbindex aus einer Palette von 1000x1000x1000 = 1 Milliarde Farben gewählt werden. Das dürfte auch in einigen Jahren der dann zur Verfügung stehenden Hardware gerecht werden.

– VST_FONT

Eine der hervorragenden Eigenschaften von GEM ist es, beliebige Zeichensätze zu laden und auf allen Geräten auszugeben. Auch Proportionalschrift ist dann möglich. Zeichensätze werden über vst_load_fonts geladen. Man erhält die Anzahl der zusätzlichen Zeichensätze, die auf einem Gerät verfügbar sind.

Standardmäßig werden zwei Zeichensätze mitgeliefert, die die Namen „SWISS“ und „DUTCH“ tragen. Bei einem Bildschirm ist außerdem der Systemzeichensatz verfügbar (wird für Menüs und Dialogboxen benutzt). Die Indizes, die bei vst_font angegeben werden müssen, um auf einen Zeichensatz zu schalten, lauten:

1 = Systemzeichensatz
2 = Swiss
3 = Swiss Thin
4 = Swiss Thin Italic
...
14 = Dutch Roman

Um also auf den Zeichensatz SWISS umzuschalten, muß

```
vst_font (vdi_handle, 2);
```

aufgerufen werden. Die beiden Sätze „SWISS“ und „DUTCH“ sind die einzigen Sätze, die auf allen Ausgabegeräten verfügbar sind. Z.B. besitzt der Drucker keinen „system font“ wie der Bildschirm. Um Daten auszutauschen, sollte also immer einer der beiden Zeichensätze gewählt werden, wenn auf eine Metadatei geschrieben wird.

Mit den Funktionen vst_color, vst_alignment und vst_effects können die Textfarbe, die Textausrichtung (linksbündig, zentriert etc.) sowie Texteffekte (kursiv oder fett) eingestellt werden.

- VSWR_MODE

Bei allen Ausgabefunktionen kann global der sogenannte Schreibmodus eingestellt werden. Er bestimmt, wie Pixel auf das Ausgabegerät gesetzt werden und ob vorhandene, gesetzte Pixel berücksichtigt werden. Vier Modi sind möglich:

- replace
- transparent
- XOR
- reverse transparent

Im Modus „replace" werden alle Pixel auf dem Ausgaberät gesetzt, wenn Pixel ausgegeben werden. Alle anderen Pixel werden zur Hintergrundfarbe. Wird z.B. eine gestrichelte Linie ausgegeben, so kommt die Hintergrundfarbe (meist weiß) an den Stellen zum Vorschein, wo die Linie unterbrochen ist. War der Bildschirm (nicht die Hintergrundfarbe) grün und ist die Strichfarbe schwarz, so entsteht eine weiß/schwarz gestrichelte Linie.

Im Modus „transparent" scheint der Hintergrund dort durch, wo die Linie unterbrochen ist. In obigem Beispiel ist die Linie grün/schwarz gestrichelt, da der Hintergund in den „Löchern" der Linie durchscheint.

Im Modus „XOR" werden einfach alle Pixel invertiert. Bei einem monochromen Bildschirm wird schwarz aus weiß und umgekehrt. Dies wird häufig gebraucht, da die zweimalige Anwendung des gleichen Zeichenbefehls das Ursprungsbild wieder herstellt.

Im Modus „reverse transparent" scheint der Hintergrund dort durch, wo die Linie nicht unterbrochen ist.

- VSF_INTERIOR
- VSF_STYLE

Mit diesen beiden Funktionen lassen sich Fülltyp und Füllmuster einstellen, die solche Grafikfunktionen wie v_bar, v_circle usw. benutzen.

Der Fülltyp (vsf_interior) kann 5 Werte annehmen:

0 = Hollow (hohl, keine Füllung)
1 = Solid (komplette einfarbige Füllung)
2 = Pattern (Füllung mit bestimmtem Muster)
3 = Hatch (Füllung mit einer Schraffur)
4 = User defined (beliebiges Füllmuster)

Wird der Fülltyp auf 2 (Pattern) oder 3 (Hatch) gestellt, so kann man mit der Funktion vsf_style zwischen 24 Mustern bzw. 12 Schraffuren wählen.

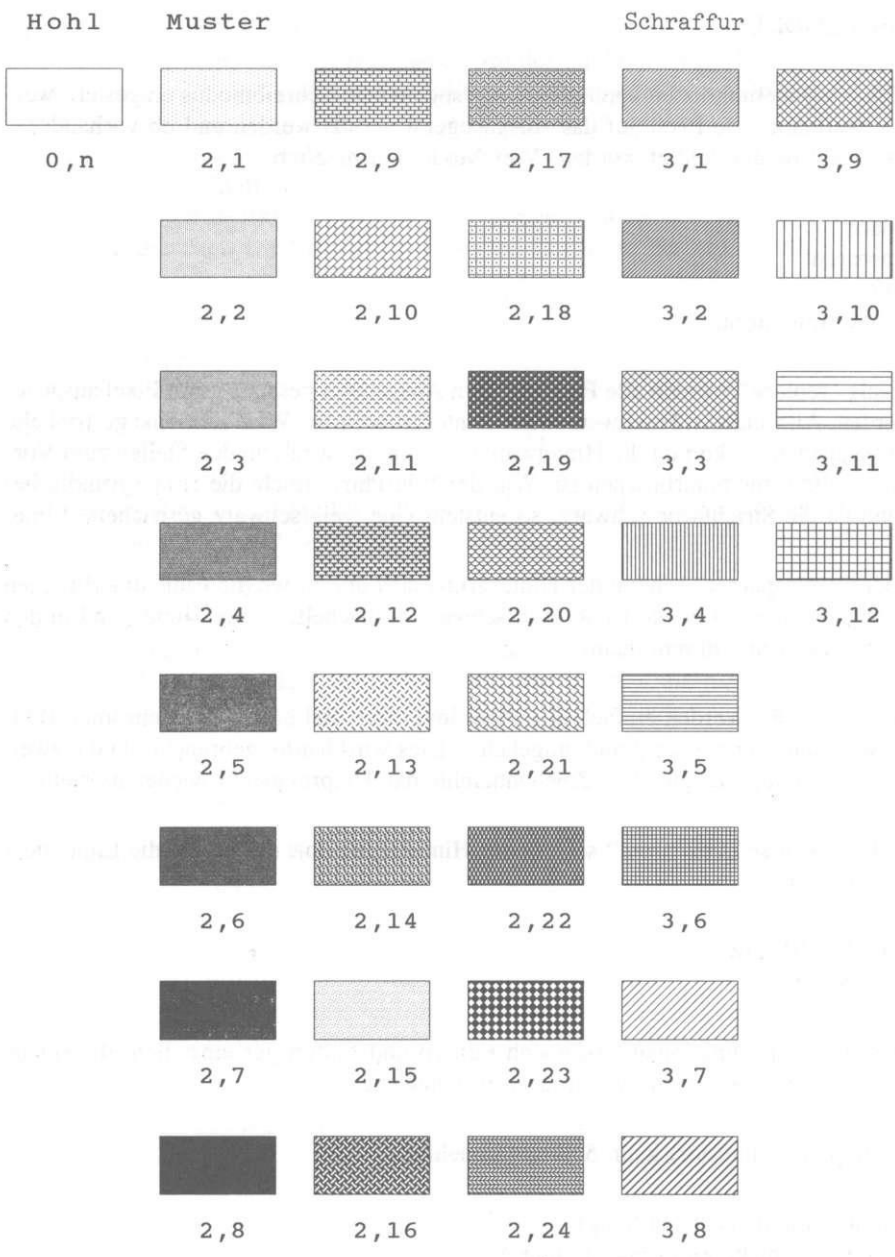



**Abb. 2.8:** Füllmuster und deren Indizes

Beispiel: Es soll die Schraffur mit Index 2 gewählt werden (siehe auch VDI.H):

```
vsf_interior (vdi_handle, FIS_HATCH);
vsf_style (vdi_handle, 2);
```

– VSF_PERIMETER

Diese Funktion hat ab GEM/3 einen Parameter dazu bekommen. Es handelt sich um den Linientyp. So können jetzt z.B. gestrichelte Umrandungen gezogen werden.

Beispiel: Setzen des Randes auf durchgezogen.

```
vsf_perimeter (vdi_handle, TRUE, SOLID);
```

Gruppe 4: Rasterfunktionen

Mit Hilfe der Rasterfunktionen ist es möglich, rechteckige Blöcke von Bits im Speicher zu manipulieren. Da der Bildschirm ebenfalls ein Bitblock ist, können die Rasterfunktionen auch für ihn benutzt werden.

| | |
|---|---|
| 105 – v_get_pixel | Holt ein Pixelelement |
| 109 – vro_cpyfm | Rasteroperation, opaque |
| 110 – vr_trnfm | Rasterblock transformieren |
| 113 – vrt_cpyfm | Rasteroperation, transparent |

Mit den Rasterfunktionen ist es auch möglich, einen Bildschirmbereich in einen anderen zu kopieren oder zu verschieben. So wird z.B. das Scrolling in einem GEM-Fenster realisiert. Man verschiebt eine Zeile Text nach oben, indem man einen rechteckigen Block nach oben verschiebt und die letzte, neu hinzugekommene Zeile neu zeichnet. Der Vorteil beim Benutzen dieser Funktionen liegt auf der Hand. Sie sind hardwareunabhängig und laufen mit jeder Grafikkarte. Grafikprozessoren werden dann automatisch unterstützt (Blitter etc.).

Bevor auf die einzelnen Funktionen eingegangen wird, muß der Begriff des „Memory Form Definition Block (MFDB)“ geklärt werden. Ein rechteckiger Pixelblock wird durch folgende Informationen beschrieben:

- Adresse des Bitblocks
- Breite und Höhe des Rasters
- Breite des Rasters in Maschinenworten zu 16 Bits
- Formatflag
- Anzahl der Farbebenen
- Werte für zukünftige Erweiterungen

Er wird durch die C-Struktur (VDI.H) wie folgt beschrieben:

```
/* Memory Form Definition Block */

typedef struct memform
{
  VOID *mp;
  WORD fwp;
  WORD fh;
  WORD fww;
  WORD ff;
  WORD np;
  WORD r1;
  WORD r2;
  WORD r3;
} MFDB;
```

Der Wert mp (memory pointer) zeigt auf den Speicherbereich des MFDB. Es kann die Startadresse eines Puffers sein oder den Wert 0 annehmen. Ist der Wert 0, so beschreibt der MFDB keinen Speicherbereich, sondern einen Bereich des physikalischen Ausgabegerätes. Öffnet man z.B. den Bildschirm als Ausgabegerät, so wird durch die Angabe von 0 im memory pointer der Bildschirm als Block beschrieben. Die anderen Werte in der Struktur braucht man dann nicht mehr anzugeben, da der Ausgabetreiber sie ja kennt (z.B. die Anzahl der Farbebenen, Breite und Höhe etc.).

Die Werte fwp (form width in pixel) und fh (form height) geben an, wie groß der Block sein soll, den man beschreiben möchte (z.B. 100x50 Pixel). Der Wert fww (form width in words) ist im Prinzip redundante Information und gibt die Breite des Blockes in 16-Bit-Worten an. In obigem Beispiel müßte er den Wert 100/16 also 6,25 haben. In diesem Falle wird aufgerundet auf 7. Auch wenn ein Block z.B. nur 3 Bit breit und 20 Bits hoch ist, so müssen 20 Worte im Speicher benutzt werden, da jede neue Pixelzeile immer auf einer Wortgrenze beginnt.

Der Wert ff (format flag) gibt an, ob sich der Block im sogenannten Standardformat oder im gerätespezifischen Format befindet. Das Standardformat ist von Digital Research fest vorgeschrieben und hat folgenden Aufbau:

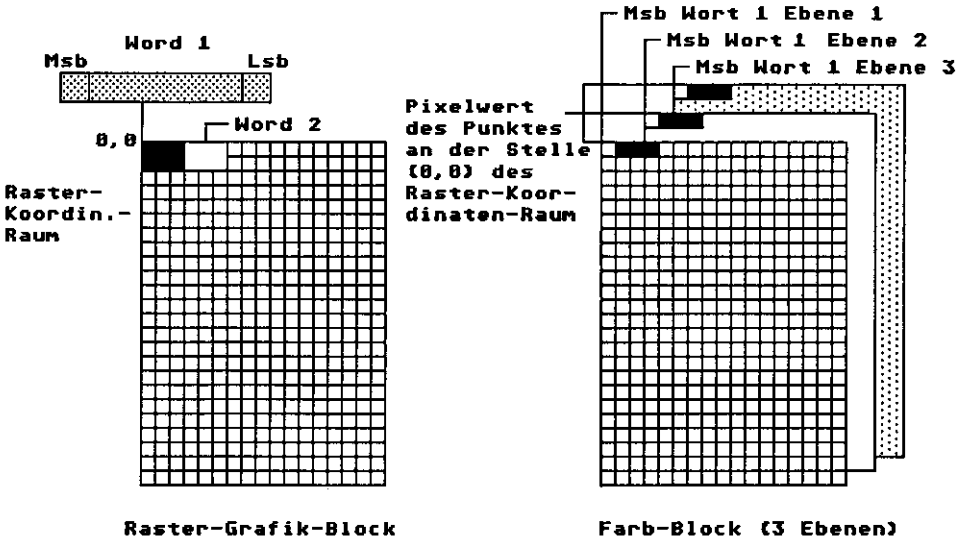



**Abb. 2.9:** Standard-MFDB

Folgende Pixelwerte für monochrome und farbige MFDBs sind vordefiniert:

a) Bildschirm mit 2 Farben (monochrom):

| Pixelwert | Farbindex | Farbe |
|---|---|---|
| 0 | 0 | weiß |
| 1 | 1 | schwarz |

b) Bildschirm mit 4 Farben:

| Pixelwert | Farbindex | Farbe |
|---|---|---|
| 00 | 0 | weiß |
| 01 | 2 | rot |
| 10 | 3 | grün |
| 11 | 1 | schwarz |

c) Bildschirm mit 8 Farben:

| Pixelwert | Farbindex | Farbe |
|---|---|---|
| 000 | 0 | weiß |
| 001 | 2 | rot |
| 010 | 3 | grün |
| 011 | 6 | gelb |
| 100 | 4 | blau |
| 101 | 7 | magenta |
| 110 | 5 | cyan |
| 111 | 1 | schwarz |

d) Bildschirm mit 16 Farben:

| Pixelwert | Farbindex | Farbe |
|---|---|---|
| 0000 | 0 | weiß |
| 0001 | 2 | rot |
| 0010 | 3 | grün |
| 0011 | 6 | gelb |
| 0100 | 4 | blau |
| 0101 | 7 | magenta |
| 0110 | 5 | cyan |
| 0111 | 8 | hellgrau |
| 1000 | 9 | dunkelgrau |
| 1001 | 10 | dunkelrot |
| 1010 | 11 | dunkelgrün |
| 1011 | 14 | dunkelgelb |
| 1100 | 12 | dunkelblau |
| 1101 | 15 | dunkelmagenta |
| 1110 | 13 | dunkelcyan |
| 1111 | 1 | schwarz |

Da auf einem ATARI ST in niedriger Auflösung 16 Farben zur Verfügung stehen, sollte man die Schieberegler im Kontrollfeld bzw. im DESKTOP.INF wie in folgender Tabelle einstellen. Nur so kann man sicher sein, daß eine Grafik, die z.B. von MS-DOS übernommen wurde, auch in den richtigen Farben erscheint. In der folgenden Tabelle wird der Farbindex, gefolgt vom RGB-Wert angegeben:

0 = 777 (weiß)
1 = 000 (schwarz)
2 = 700 (rot)

3 = 070 (grün)
4 = 007 (blau)
5 = 077 (cyan)
6 = 770 (gelb)
7 = 707 (magenta)
8 = 666 (hellgrau)
9 = 333 (dunkelgrau)
10 = 500 (dunkelrot)
11 = 050 (dunkelgrün)
12 = 005 (dunkelblau)
13 = 055 (dunkelcyan)
14 = 550 (dunkelgelb)
15 = 505 (dunkelmagenta)

Wie man sieht, gehört zu einem Monochrombildschirm eine Farbebene, in dem die Werte im Puffer des MFDBs die beiden Farben Weiß und Schwarz beschreiben. Jedes Bit, das im Puffer den Wert 0 hat, wird weiß dargestellt, während ein gesetztes Bit die Farbe Schwarz erzeugt. Das höchste Bit des ersten Wortes im Puffer bestimmt also die Farbe des Punktes (0,0), das zweithöchste Bit die Farbe von (1,0) und das niederwertigste Bit die Farbe des Punktes (15,0). Um also den linken Punkt (0,0) auf Schwarz zu setzen, muß der Puffer folgenden Wert beinhalten:

```
buffer [0] = 0x8000; /* = 1000000000000000 in Bit-Schreibweise */
```

Ähnlich funktioniert es mit einer Farbgrafik. Um bei einem 8-Farben-Schirm den Punkt (0,0) auf Magenta zu setzen, muß der Pixelwert 101 in die einzelnen Farbebenen eingetragen werden. Bei einer Farbebene der Länge 100 Bytes (s. Beispiel unten) wäre die Zuweisung:

```
buffer [0]   = 0x8000; /* = 1000000000000000 */
buffer [100] = 0;      /* = 0000000000000000 */
buffer [200] = 0x8000; /* = 1000000000000000 */
```

Die Anzahl der Farbebenen kann man durch den VDI-Aufruf vq_extnd bestimmen (work_out [4]).
Da es sehr viele Rechner und Grafikkarten auf dem Markt gibt, weiß ein GEM-Programmierer nie, wie der Bildschirmspeicher des gerade angeschlossenen Ausgabegerätes intern aufgebaut ist. Würde er direkt ins Video-Ram schreiben, so würde sein Programm auf anderen Rechnern oder Grafikkarten abstürzen oder andere unerwartete Effekte zeigen.
Zu diesem Zweck wurde ein Standardformat vorgeschlagen. Das VDI stellt eine Funktion (vr_trnfm) zur Verfügung, mit der man eine Umwandlung vom Standardformat in das Geräteformat vornehmen kann.

Möchte man z.B. eine Farbgrafik mit 80x20 Pixel und 8 Farben erzeugen und weiß nicht, wie der Bildschirm intern aufgebaut ist, so legt man sich im Speicher einen Bereich an, in dem das Bild aufgebaut wird. Dieser Speicherbereich müßte 100 = 80/16x20 Bytes

pro Farbebene groß sein. Da man 8 = 2^3 Farben haben möchte, braucht man 3 Ebenen (planes), also 300 Bytes. Die Belegung des MFDB sähe dann wie folgt aus:

```
MFDB mfdb;
BYTE buffer [300];

mfdb.mp  = (VOID *)buffer;
mfdb.fwp = 80;
mfdb.fh  = 20;
mfdb.fww = 5;
mfdb.ff  = 1;
mfdb.np  = 3;
```

Der Puffer enthält die Informationen, wie in Abb. 2.9 beschrieben. Die Farbgrafik kann auf den Bildschirm gebracht werden, indem man zuerst die VDI-Transformationsfunktion aufruft. Dadurch wird die komplette Grafik an die Grafikkarte angepaßt. Anschließend kann dann mit der Funktion vro_cpyfm der gesamte Block auf den Bildschirm geschrieben werden.

Noch ein Beispiel: Piktogramme in Resource-Dateien befinden sich immer im Standardformat. Wird eine Dialogbox, die Piktogramme enthält, mit der AES-Funktion objc_draw auf den Bildschirm gezeichnet, so werden die Piktogramme (kleine MFDBs) vorher nicht automatisch transformiert, sondern direkt in den Bildschirm kopiert. Bei einem ATARI ST ist dies nicht weiter tragisch, da das Standardformat genau dem gerätespezifischen Format entspricht.

Wer aber GEM-Software auf MS-DOS portiert, erlebt sein „erstes" Wunder, wenn Piktogramme z.B. auf einer EGA-Karte gezeichnet werden. Sie werden invers und verdreht dargestellt, da die EGA-Karte nicht dem Standardformat entspricht. Wandelt man die Piktogramme einer RSC-Datei aber mittels vr_trnfm um, so erscheinen die Piktogramme wieder richtig. Auf einem ATARI ST kann diese Funktion ruhig aufgerufen werden, da sie keine Änderungen der Piktogramme vornimmt. Wie dies genau gemacht wird, erkennt man sofort an dem Modul RESOURCE, das auf der Begleitdiskette enthalten ist.

Wenn man Rasteroperationen vornimmt, so kann man auch noch angeben, wie zu setzende Pixel mit den schon vorhandenen Pixeln logisch verknüpft werden sollen. Die Konstanten befinden sich in der Datei VDI.H (bit blt rules) und sollten benutzt werden. Sie beginnen mit ALL_WHITE (0) und enden mit ALL_BLACK (15). Dabei bedeutet S die Quelle (source) und D das Ziel (destination).

– VRO_CPYFM

Dies ist die wichtigste Funktion aus der Raster-Gruppe. Mit ihr können rechteckige Bildausschnitte kopiert (verschoben) werden, wobei eine logische Operation angegeben wird. Auch Bildschirmbereiche können in einen Speicherbereich und umgekehrt verschoben

werden (z.B. für das Retten des Hintergrundes). Das hardwareunabhängige Scrollen in Fenstern wird mit Hilfe dieser Funktion erst möglich. Außerdem übergibt man der Funktion zwei Rechtecke, die die Quelle und das Ziel auf dem Ausgabegerät bestimmen. Wenn die beiden angegebenen MFDBs gleich sind (z.B. Kopieren von Bildschirm nach Bildschirm), so spielt die Angabe der Rechtecke keine Rolle. D.h. auch überlappende Rechtecke von Quelle und Ziel werden korrekt kopiert. Die beiden angegebenen MFDBs müssen sich beide im gerätespezifischen Format befinden.

Sind Quellen- und Zielrechteck nicht gleich groß, so wird in die Adresse des Zieles kopiert und das Quellraster entsprechend verkleinert oder vergrößert. Rastervergrößerungen bzw. -verkleinerungen sind also mit einem Aufruf zu machen. Allerdings muß der Treiber des Ausgabegerätes dies unterstützen. Zur Abfrage kann die VDI-Funktion vq_extnd (work_out [3]) benutzt werden.

– VR_TRNFM

Wie weiter oben erwähnt wurde, gibt es zwei Formate: Das Standardformat und das gerätespezifische Format. Standardformate werden u.a. in Resource-Dateien für Icons und Bit-Images benutzt.
Bevor diese auf den Bildschirm ausgegeben werden können, müssen sie in das Format des Ausgabegerätes überführt werden.

Das VDI gibt beim Zeichnen von Icons und Bit-Images in Dialogboxen nur noch die reinen Pixeldaten auf den Bildschirm aus. Dadurch spart es sich viel Zeit, da Piktogramme meist öfters gezeichnet werden sollen. Das bedeutet, daß sich der Programmierer darum kümmern muß, die Daten aus den Piktogrammen und Zeichnungen vom Standardformat in das Geräteformat zu transformieren. Dies übernimmt aber glücklicherweise die Funktion vr_trnfm, die lediglich pro Piktogramm oder Zeichnung einmal aufgerufen werden muß. Sie invertiert Bits und vertauscht die Bit-Reihenfolge eines Blocks, falls das Ausgabegerät dies benötigt. Man übergibt ein Flag, in welchem Format sich der Quell-Bit-Block befindet, und die Funktion wandelt den kompletten Block in den Ziel-Block um, wobei sie das Flag invertiert.

Die beiden Blöcke müssen entweder exakt identisch sein oder komplett verschieden. Überlappungen sind nicht erlaubt. So ist es möglich, einen Standardformat-Block und einen gerätespezifischen Block gleichzeitig zu handhaben oder den Block in sich selbst umzuwandeln. Das Umwandeln in sich selbst wird im Modul RESOURCE auf der beiliegenden Diskette behandelt. Alle Piktogramme und Bitblöcke werden vom Standardformat ins gerätespezifische Format umgewandelt.

– VRT_CPYFM

Über diese Funktion wurde in keinem Buch ausführlicher gesprochen. Der Grund liegt wahrscheinlich daran, daß sie nie richtig verstanden wurde. Man stelle sich vor,

man erzeuge Piktogramme im Resource-Construction-Set. Diese Piktogramme sind monochrome Grafikblöcke, also mit nur 1 Farbebene. Die Tatsache, daß man Piktogrammen eine Farbe geben kann, spielt keine Rolle, da sie ja nicht mehrfarbig sein können. Nun möchte man diese Piktogramme nicht über eine Dialogbox vom AES gezeichnet haben, sondern möchte sie selbst benutzen.

Wenn ein Farbbildschirm angeschlossen ist, so wird der Befehl vro_cpyfm keine Hilfe sein, da das Quellraster 1 Farbebene, das Zielraster aber 4 Farbebenen (bei 16 Farben) besitzt. Hier greift nun die Funktion vrt_cpyfm ein. Mit ihr ist es möglich, ein einfarbiges Quellraster in ein mehrfarbiges Zielraster zu kopieren. Man gibt die beiden Farben an, die für gesetzte und nicht gesetzte Pixel aus dem Quellraster benutzt werden sollen.

Ohne diese Funktion wäre es nicht möglich, Piktogramme bzw. Bitblöcke für benutzerdefinierte Objekte zu gebrauchen. Solche Objekte können nur mit Hilfe des VDI gezeichnet werden. Ein Beispiel befindet sich im Modul RESOURCE oder im Programm SHOWGEM, in dem sogenannte (Macintosh-)Radio-Buttons (rund mit schwarzem Kern) in Dialogboxen gezeichnet werden.

– Gruppe 5: Eingabefunktionen

Diese Funktionen werden vom AES benutzt, um die Maus zu steuern und um Tastatureingaben vornehmen zu können. Da sie aber von einem GEM-Anwendungsprogramm nicht benutzt werden dürfen, werden sie nur der Vollständigkeit halber aufgeführt. Lediglich Programme, die nur auf dem VDI aufsetzen, können Nutzen aus ihnen ziehen.

| | | |
|---|---|---|
| 28 | vrq_locator | Input locator, Anfragemodus |
| 28 | vsm_locator | Input locator, Testmodus |
| 29 | vrq_valuator | Input valuator, Anfragemodus |
| 29 | vsm_valuator | Input valuator, Testmodus |
| 30 | vrq_choice | Input choice, Anfragemodus |
| 30 | vsm_choice | Input choice, Testmodus |
| 31 | vrq_string | Input string, Anfragemodus |
| 31 | vsm_string | Input string, Testmodus |
| 33 | vsin_mode | Eingabemodus wählen |
| 111 | vsc_form | Mausform setzen |
| 118 | vex_timv | Timer Interrupt verändern |
| 122 | v_show_c | Cursor zeigen |
| 123 | v_hide_c | Cursor verstecken |
| 124 | v_qmouse | Maustasten abfragen |
| 125 | vex_butv | Mausknopfvektor verändern |
| 126 | vex_motv | Mausbewegungsvektor verändern |
| 127 | vex_curv | Cursorzeichenvektor verändern |
| 128 | vq_key_s | Tastaturstatus holen |

– VSIN_MODE

Mit Hilfe dieser Funktion kann das Eingabegerät und dessen Funktionsweise eingestellt werden. Eingabegeräte können sein:

a) locator (Maus, Trackball, Joystick)
b) valuator (Potentiometer)
c) choice (Funktionstasten)
d) string (Tastatur)

Die Funktionsweise kann auf Request oder Sample geschaltet werden. Im Request-Modus wartet der Eingabetreiber, bis ein Eingabeereignis vorliegt und kehrt dann erst zurück. Im Sample-Modus wird der Status ohne zu warten zurückgegeben.

– VEX_TIMV
– VEX_BUTV
– VEX_MOTV
– VEX_CURV

Zu diesen Funktionen soll gesagt werden, daß sie nur von den Applikationen verwendet werden können, die die physikalische Arbeitsstation geöffnet haben. Sie sind also für GEM-Applikationen unbrauchbar.

– Gruppe 6: Nachfragefunktionen

Um GEM-Programme möglichst unabhängig von der zugrundeliegenden Hardware zu schreiben, muß es Funktionen geben, mit denen man alle Informationen über die angeschlossenen Geräte erfragen kann. Sowohl die Nachfrage als auch die Escape-Funktionen erfüllen diesen Zweck.

| | | |
|---|---|---|
| 26 – | vq_color | Farbeinstellungen erfragen |
| 27 – | vq_cellarray | Zellenraster erfragen |
| 35 – | vql_attributes | Linienattribute erfragen |
| 36 – | vqm_attributes | Polymarkerattribute erfragen |
| 37 – | vqf_attributes | Füllflächenattribute erfragen |
| 38 – | vqt_attributes | Textattribute erfragen |
| 102 – | vq_extnd | Weitere Geräteinformationen holen |
| 115 – | vq_inmode | Eingabemodus erfragen |
| 116 – | vqt_extent | Textrahmen erfragen |
| 117 – | vqt_width | Zeichengröße erfragen |
| 130 – | vqt_name | Zeichensatzname und Index erfragen |
| 131 – | vqt_font_info | Zeichensatzinformationen erfragen |
| 132 – | vqt_justified | Informationen über ausgerichteten Text erfragen |

Die meisten der o.g. Funktionen erklären sich von selbst und sind in anderen Büchern ausreichend beschrieben. Wir werden daher nur noch auf die Funktionen eingehen, bei denen die meisten Schwierigkeiten bestehen bzw. die zum Verständnis notwendig sind.

– VQ_EXTND

Hiermit können weitere Geräteinformationen geholt werden. Interessant sind hier folgende Werte:

work_out [3]: Skalierung von Rasterblöcken
work_out [4]: Anzahl der Farbebenen

Eine 1 sagt aus, daß man bei der Funktion vro_cpfm von einem Rechteck aus in ein kleineres oder größeres Rechteck kopieren kann und daß das VDI die Verkleinerung/Vergrößerung selbst vornimmt. Bei pixelorientierten Zeichenprogrammen kann dies eine sehr große Hilfe darstellen. Im übrigen werden Zeichensätze auf Bildschirmen mit der Funktion vst_height ebenfalls beliebig skaliert.

Ab GEM/3 werden die Felder work_out [20] bis work_out [27] wie folgt benutzt:

work_out [20]: Erweiterte Informationen zur Pixelgröße in Micrometer,
diese werden dann in work_out [21] und work_out [22]
zurückgegeben
0 = keine weiteren Informationen vorhanden
1 = Pixelgröße in 1/10000 mm
2 = Pixelgröße in 1/100000 mm
3 = Pixelgröße in 1/1000000 mm
work_out [21]: Punktbreite in Einheiten aus work_out [20]
work_out [22]: Punkthöhe in Einheiten aus work_out [20]
work_out [23]: Punkte/Zoll in x-Richtung
work_out [24]: Punkte/Zoll in y-Richtung
work_out [25]: Flag, um anzuzeigen, ob Bit-Image-Dateien auf Druckern
vom Treiber gedreht werden können.
0 = Nein
1 = Drehungen um 0, 90, 180 und 270 Grad erlaubt
work_out [26] – work_out [27]: Adresse des Bildschirmpuffers, der bei
AES-Funktionen benutzt wird (1/4 des Gesamtschirms).

– VQT_EXTENT

Hiermit kann zu einer gegebenen Zeichenkette ein Rahmen erfragt werden, der genau um die Zeichenkette paßt. Alle Zeichenattribute werden berücksichtigt. Man erspart sich damit das Ausrechnen, ob eine Zeichenkette noch in einen bestimmten Bereich einer Grafik hineinpaßt. Zurückgegeben werden 4 Punkte (relativ, keine Bildschirmkoordinaten),

die Eckpunkte der Zeichenkette, beginnend links unten. Leider funktioniert dies auf einem ATARI ST nicht, wenn die Zeichenkette Umlaute enthält, da die alte GEM-Version mit Umlauten noch nicht umgehen konnte.

– VQT_NAME

Mit der Funktion vst_load_fonts (s.o.) konnten externe Zeichensätze nachgeladen werden. Dabei war der Funktionswert die Anzahl der zusätzlich verfügbaren Zeichensätze. Dieser Wert (z.B. 2) wird Elementnummer genannt. Existieren also 2 zusätzliche Sätze, so hat man auf einem Bildschirm insgesamt 3 Zeichensätze zur Verfügung: der Systemsatz und die beiden geladenen. Über die Funktion vst_font kann einer der drei Sätze gewählt werden. Der Eingabeparameter war aber ein Index, der zu dem Font gehört. Bisher hatten wir keine Möglichkeit, diesen Index zu bestimmen.

Hier greift die Funktion vqt_name ein. Sie bestimmt zu einer Elementnummer den Index für vst_font, sowie den Namen und den Stil des angegebenen Zeichensatzes. Namen können sein: „System font", „Swiss", „Dutch". Ein Stil kann sein: „Light", „Italic" usw. Eine C-Beispielaufruffolge könnte so aussehen:

```
WORD index, element_num, num_fonts;
WORD table [11]; /* für maximal 10 Fonts */
BYTE name [32];

num_fonts = vst_load_fonts (vdi_handle, 0) + 1;
for (element_num = 1; element_num <= num_fonts; element_num++)
{
  index = vqt_name (vdi_handle, element_num, name);
  table [element_num] = index;
} /* for */

vst_font (vdi_handle, table [3]);
```

Wir nehmen an, daß es zwei weitere Zeichensätze gibt, die „Swiss" und „Dutch" heißen. In obigem Beispiel werden zunächst die Anzahl der insgesamt zur Verfügung stehenden Fonts ermittelt. Dieser Wert (3) wird in num_fonts eingetragen. Das Beispiel gilt nur für den Bildschirm, da es hier immer einen Zeichensatz mehr gibt (+1). In der nachfolgenden Schleife wird zu allen 3 Sätzen der Index besorgt. Man übergibt alle Nummern von 1 bis 3 an die Funktion vqt_name und erhält die Indizes, die in dem Feld table abgelegt werden. Die Indizes, die man bei vqt_name erhält, werden 1 (System font), 2 (Swiss) und 14 (Dutch) sein. Nach der Schleife wählen wir den Font „Dutch" aus. Es ist der dritte Font und an vst_font wird der Wert 14 = table [3] übergeben.

– Gruppe 7: Escapefunktionen

Die sogenannten Escapefunktionen dienen dazu, spezielle Eigenschaften und Möglichkeiten von grafischen Ausgabegeräten auszunutzen. Sie haben alle die Funktionsnummer 5 und eine Unterfunktionsnummer (subopcode). In nachfolgender Tabelle werden nur die Unterfunktionsnummern angegeben.

| | | |
|---|---|---|
| 1 | vq_chcells | Alphanumerische Cursorabfragen |
| 2 | v_exit_cur | Auf Grafikmodus schalten |
| 3 | v_enter_cur | Auf Textmodus schalten |
| 4 | v_curup | Textcursor eine Zeile nach oben setzen |
| 5 | v_curdown | Textcursor eine Zeile nach unten setzen |
| 6 | v_curright | Textcursor eine Spalte nach rechts setzen |
| 7 | v_curleft | Textcursor eine Spalte nach links setzen |
| 8 | v_curhome | Textcursor auf Position 0,0 bringen |
| 9 | v_eeos | Lösche bis zum Ende des Bildschirms |
| 10 | v_eeol | Lösche bis zum Ende der Zeile |
| 11 | vs_curaddress | Setze Textcursor an bestimmte Position |
| 12 | v_curtext | Text an Cursorposition ausgeben |
| 13 | v_rvon | Inversschrift einschalten |
| 14 | v_rvoff | Inversschrift ausschalten |
| 15 | vq_curaddress | Position des Textcursors erfragen |
| 16 | vq_tabstatus | Status eines Grafiktabletts holen |
| 17 | v_hardcopy | Bildschirm auf Drucker ausgeben |
| 18 | v_dspcur | Grafikcursor an best. Stelle setzen |
| 19 | v_rmcur | Grafikcursor löschen |
| 20 | v_form_adv | Seitenvorschub ausgeben |
| 21 | v_output_window | Teil eines Bildes auf Drucker ausgeben |
| 22 | v_clear_disp_list | Grafikliste des Druckers löschen |
| 23 | v_bit_image | Bit-Image-Dateien drucken |
| 24 | vq_scqan | Spezifikation des Druckkopfes erfragen |
| 25 | v_alpha_text | Alphanumerischen Text auf Drucker geben |
| 27 | v_orient | Hoch- oder Querformat einstellen |
| 28 | v_copies | Anzahl Kopien einstellen |
| 29 | v_tray | Papierschacht bestimmen |
| 60 | vs_palette | Farbpalette einstellen |
| 61 | v_sound | Sound ausgeben |
| 62 | vs_mute | Sound ein-/ausschalten |
| 81 | vt_resolution | Auflösung des Grafiktabletts in Zeile/Zoll |
| 82 | vt_axis | Auflösung des Grafiktabletts einstellen |
| 83 | vt_origin | Ursprung für Grafiktablett einstellen |
| 84 | vq_tdimensions | Dimensionen des Grafiktabletts bestimmen |
| 85 | vt_alignment | Ausrichtung des Grafiktabletts einstellen |
| 91 | vsp_film | Kamerafilmtyp und -belichtung einstellen |
| 92 | vqp_filmname | Filmnamen für Kameras erfragen |

| | | |
|---|---|---|
| 93 | – vsc_expose | Belichtung bei Preview ein-/ausschalten |
| 98 | – v_meta_extents | Ausdehnung der Grafik in Metadatei |
| 99 | – v_write_meta | Schreibe Element in Metadatei |
| 100 | – vm_filename | Änderung des Dateinamens der Metadatei |
| 101 | – v_xbit_image | Erweiterte Funktion von v_bit_image |

Viele der o.g. Funktionen werden nie benutzt, wie z.B. die Funktionen für die Ausgabe auf einen Film oder das Benutzen eines Grafiktabletts. Sie werden auch nicht weiter erläutert, da wir keine Filmbelichtungsanlage zum Testen zur Verfügung hatten. Wir werden daher nur noch auf die Funktionen eingehen, die benutzt werden sollten und die auch vorhanden sind.

Die Funktionen 1 – 15 (vq_chcells bis vq_curaddress) beziehen sich alle auf die Ausgabe des alphanumerischen Bildschirms. In der Tat kann auch auf einem ATARI ST, der ständig im Grafikmodus arbeitet, auf den alphanumerischen Schirm umgeschaltet werden (v_enter_cur). Der Bildschirm wird dabei automatisch gelöscht und Textausgaben können gemacht werden. Z.B. benutzt der VT52-Emulator (ein Accessory) dieses Feature.

– V_HARDCOPY

Hiermit kann der Bildschirminhalt auf einen Drucker oder ein anderes Gerät ausgegeben werden. Der Druckertreiber muß diese Funktion allerdings zur Verfügung stellen.

– V_DSPCUR
– V_RMCUR

Diese beiden Funktionen werden vom GEM-System bei der physikalischen Eröffnung des Ausgabegerätes Bildschirm bzw. beim Wechseln vom Grafik- auf Textmodus benutzt. Der Mauscursor wird beim Start des Systems immer zuerst in der Mitte des Bildschirms sichtbar. Auch ein Umschalten von Text- auf Grafikmodus (v_exit_cur) läßt ihn wieder in der Mitte erscheinen, wenn es einen „richtigen" Text- und Grafikmodus gibt (nicht bei ATARI ST).

– V_FORM_ADVANCE

Durch Aufruf von v_form_advance kann zwischenzeitlich ein Seitenvorschub auf dem Drucker ausgegeben werden. Der Seitenvorschub wird sofort ausgeführt. Bei der Funktion v_clrwk wird ebenfalls ein Seitenvorschub ausgegeben, aber die Grafik, die inzwischen ausgegeben wurde, wird hierbei gelöscht.

– V_OUTPUT_WINDOW

Grafiken, die auf einen Drucker ausgegeben werden, werden vom VDI zunächst gepuffert. Erst bei der Funktion v_updwk wird der Puffer geleert und die Grafik tatsächlich zu Papier gebracht. Falls man sich vor dem Drucken entschließt, doch nur einen Teil des Bildes auszugeben, so kann statt v_updwk auch die Funktion v_output_window verwendet werden. Man gibt ein Fensterrechteck an, das bestimmt, welche Teile der Grafik ausgegeben werden sollen.

– V_CLEAR_DISP_LIST

Der Puffer, in dem die Grafikbefehle für einen Drucker gespeichert werden, wird gelöscht, wobei kein Seitenvorschub ausgegeben wird. Diese Funktion kann statt v_clrwk benutzt werden, bei der immer ein Seitenvorschub ausgegeben wird. Ein Beispiel wäre das Unterbrechen einer laufenden Grafikausgabe mit Löschen des Grafikpuffers.

– V_ALPHA_TEXT

Mit Hilfe dieser Funktion ist es möglich, alphanumerischen Text auf einem Drucker auszugeben. Die angegebene Zeichenkette wird nicht im Grafikmodus ausgegeben, sondern im Standard-Druckerzeichensatz. Das WORDPLUS, das mit GEM/3 mitgeliefert wird, bedient sich dieser Methode, um Texte auszugeben. Es existieren keine Konfigurationsdateien mehr für den Drucker (HEX, CFG), vielmehr erzeugt WORDPLUS eine OUT-Datei (siehe 2.7.3), in der alle Informationen für den Text stehen. Bei der Ausgabe des Textes auf die OUT-Datei werden außerdem Steuerzeichen mitgeschrieben, die die Attribute eines Textes (fett, kursiv, unterstrichen etc.) bestimmen.

Diese Steuerzeichen für Textattribute sind von Digital Research vorgeschrieben und werden vom OUTPUT an den Drucker weitergeleitet. Es ist Aufgabe des Druckertreibers, diese zu analysieren und auf das entsprechende Textattribut umzuschalten. Alle Steuerzeichen werden mit dem ASCII Code DC2 (Dezimalwert 18) eingeleitet. Folgende Steuerzeichen sind definiert und schalten ein Attribut ein oder aus:

(DC2)0 – Fettschrift ein
(DC2)1 – Fettschrift aus
(DC2)2 – Kursivschrift ein
(DC2)3 – Kursivschrift aus
(DC2)4 – Unterstreichen ein
(DC2)5 – Unterstreichen aus
(DC2)6 – Superscript ein
(DC2)7 – Superscript aus
(DC2)8 – Subscript ein
(DC2)9 – Subscript aus
(DC2)A – NLQ-Schrift ein

(DC2)B – NLQ-Schrift aus
(DC2)C – Breitschrift ein
(DC2)D – Breitschrift aus
(DC2)E – Hellschrift ein
(DC2)F – Hellschrift aus
(DC2)G – (DC2)V – reserviert
(DC2)W – Pica ein
(DC2)X – Elite ein
(DC2)Y – Schmalschrift ein
(DC2)Z – Proportionalschrift ein

Mit Hilfe dieser Steuerzeichen wäre es möglich, Texte mit Attributen zwischen verschiedenen Programmen auszutauschen. Die Textzeile „Dies ist **fett** bzw. *kursiv*." müßte dann so aussehen:

Dies ist (DC2)0fett(DC2)1 bzw. (DC2)2kursiv(DC2)3.

Eine Textdatei, nennen wir sie „TEST.OUT", die o.g. Format hat, wird tatsächlich von GEM-OUTPUT verstanden und korrekt auf dem Drucker ausgegeben.

– V_ORIENT

Diese GEM/3-Funktion wird dazu benutzt, um das Ausgabegerät auf eine der beiden Ausgabemöglichkeiten Hoch- oder Querformat (Portrait bzw. Landscape) einzustellen. Die Einstellung muß vorgenommen werden, bevor irgendwelche Ausgaben getätigt werden.

Beispiel: Möchte man auf den Drucker eine Grafik im Querformat ausgeben, so muß wie folgt vorgegangen werden (siehe auch SHOWGEM.C):

```
vdi_handle = open_work (PRINTER);
v_orient (vdi_handle, OR_LANDSCAPE);
/* Grafikausgaben... */
```

– V_COPIES

Mit dieser GEM/3-Funktion kann die Anzahl der Kopien, die auf einem Drucker gemacht werden sollen, eingestellt werden. Alle Seiten bis zum Schließen des Arbeitsgerätes werden in der spezifizierten Anzahl ausgegeben.

C-Aufruf:

```
count = 3;
v_copies (vdi_handle, count);
```

– V_TRAY

Moderne Drucker können mehrere Einzugsschächte für Einzelpapier besitzen. Papier muß manchmal aber auch manuell eingezogen werden. Die GEM/3-Funktion v_tray bestimmt den Einzugsschacht, aus welchem das Papier geholt werden soll. Dabei gelten folgende Konstanten:

– 1: Manueller Einzug
  0: Default Einzug
  1: Erster optioneller Einzugsschacht
  n: n-ter optioneller Einzugsschacht

Beispiel: Papier soll manuell eingezogen werden (siehe auch VDI.H):

```
vdi_handle = open_work (PRINTER);
v_tray (vdi_handle, TRAY_MANUAL);
```

– V_SOUND

Diese GEM 2.X-Funktion kann benutzt werden, um Töne aus dem Lautsprecher auszugeben. Dabei übergibt man die Frequenz und die Länge des Tones.

Beispiel:

```
frequency = 550;
duration = 3;
v_sound (vdi_handle, frequenzy, duration);
```

– VS_MUTE

Diese GEM 2.X-Funktion wird benutzt, um den Sound von v_sound ein- und ausschalten zu können. Außerdem kann man den Status des Soundflags abfragen. Diese Aktion kann folgende Werte annehmen:

– 1: Hole Status des Soundflags
  0: Sound einschalten
  1: Sound ausschalten

Beispiel: Sound einschalten

```
action = 0;
vs_mute (vdi_handle, action);
```

- VSP_FILM
- VQP_FILMNAME
- VSC_EXPOSE

Diese 3 GEM 2.X-Funktionen ersetzen die 6 alten Funktionen für die Filmbelichtung (vqp_films, vqp_state, vsp_state, vsp_save, vsp_message, vqp_error) aus den GEM 1.X-Funktionen. Die neuen Funktionen sollen nur kurz erläutert werden.

Mit vsp_film wird der Filmtyp sowie die Belichtungszeit eingestellt. Der Aufruf lautet:

```
vsp_film (vdi_handle, index,, lightness);
```

Mit Index wird der Filmtyp angegeben. Man erhält die gültigen Indizes (1..n) über die Funktion vqp_filmname. Die Variable lightness kann Werte von −3 bis 3 annehmen. Jeder Wert ändert die Belichtung um 1/3. D.h. mit dem Wert −3 wird der Film zur Hälfte des normalen Wertes belichtet, während der Wert 3 doppelte Belichtungszeit bedeutet.

Die Funktion vqp_filmname wird verwendet, um die Namen der Filme zu erhalten, die belichtet werden können. Der Aufruf lautet:

```
BOOLEAN status;
WORD    index;
BYTE    name [25];
status = vqp_filmname (vdi_handle, index, name);
```

In einer Schleife wird man alle Filmnamen erfragen, bis die Variable status den Wert FALSE ergibt. In diesem Fall ist der Inhalt von name die leere Zeichenkette.

Beispiel: Gib alle verfügbaren Filmnamen aus

```
status = TRUE;
index  = 1;

while (status == TRUE)
{
  status = vqp_filmname (vdi_handle, index, name);
  printf ("Film [%d] = %s\n", index, name);
  index++;
} /* while */
```

Die Funktion vsc_expose schaltet die Belichtung ein oder aus. Bei manchen Kameras kann ein Preview ohne Belichtung durchgeführt werden.

Aufruf:

```
vsc_expose (vdi_handle, state);
```

Die Variable state kann folgende Werte annehmen:

0: Belichtung ausschalten
1: Belichtung einschalten

- V_META_EXTENTS
- V_WRITE_META
- VM_FILENAME
- VM_PAGESIZE
- VM_COORDS

Die Funktion v_meta_extents wird dazu benutzt, um in einer Metadatei ein Rechteck anzugeben, das die gesamte Grafik umgibt. Das Rechteck wird durch die beiden gegenüberliegenden Punkte min_x, min_y sowie max_x und max_y angegeben. So kann ein Zeichenprogramm, das Metadateien einliest, schnell feststellen, welche Ausdehnung die in der Metadatei gespeicherte Grafik besitzt. Die Werte werden in den Kopf der Metadatei eingetragen (s. 2.7.1).

Beispiel: Eine Metadatei soll eine Grafikfläche von $10000 \times 10000$ Punkten darstellen. In dieser Fläche befindet sich eine Linie mit den Koordinaten $x1 = 200$, $y1 = 400$, $x2 = 600$, $y2 = 200$. Die Ausdehnung der gesamten Grafik (hier also nur der Linie) beträgt

```
min_x = 200
min_y = 200
max_x = 600
max_y = 400
```

Der VDI-Aufruf v_meta_extents (vdi_handle, min_x, min_y, max_x, max_y) trägt in den Kopf der Metadatei die 4 Werte ein. Das Programm DUMPMETA (s. Kap. 2.7.1) gibt eine Metadatei in lesbarer Form aus. Ein weiteres Beispiel kann dem Programm VDITEST, das weiter oben schon vorgestellt wurde, entnommen werden.

Die Funktion v_write_meta wird dazu benutzt, um Informationen in die Metadatei zu schreiben, die nicht durch VDI-Funktionen abgedeckt sind. Z.B. kann GEMDRAW mehrere Objekte (Ellipsen, Rechtecke etc.) zu einem Objekt zusammenfassen. Würde es eine Ellipse gefolgt von einem Rechteck in die Metadatei schreiben und sie dann schließen, so könnte es bei erneutem Einlesen der Datei nicht mehr sagen, daß die Ellipse und das Rechteck zu einem Objekt zusammengefaßt wurden.

Dies kann dadurch verhindert werden, daß GEMDRAW zusätzliche Informationen in die Metadatei schreibt, um diese Information später wieder zu verwenden. Wird solch eine Metadatei mit Zusatzinformationen z.B. über OUTPUT ausgegeben, so stört es den Ausgabetreiber (z.B. den Drucker) nicht, daß diese Informationen vorhanden sind. Solche Informationen werden nämlich von allen Treibern einfach ignoriert.

Es können also beliebige Zusatzinformationen in die Metadatei aufgenommen werden. Diese Informationen können Integerwerte (intin) und Punktepaare (ptsin) sein. Im Feld intin [0] muß sich ein benutzerdefinierter Subopcode befinden, der die Zusatzinformation kenntlich macht. Die Nummern 0 bis 100 sind vom VDI (OUTPUT, GEMDRAW) reserviert. Eigene Informationen müssen also mit der Nummer 101 beginnen. Das Programm DUMPGEM berücksichtigt alle bekannten Subopcodes von 0 bis 100 und gibt sie im Klartext aus.

Um zu verhindern, daß es Kollisionen gibt (verschiedene Applikationen benutzen gleiche Subopcodes für verschiedene Informationen), schlagen wir vor, als erstes Objekt einer Metadatei ein Element mit Subopcode 101 zu schreiben, das den Namen und die Versionsnummer der Applikation angibt, die die Metadatei erzeugt hat. Eine andere Applikation, die die Metadatei einliest, kann feststellen, ob es sich um eine fremde oder eine eigene Metadatei handelt und alle weiteren Subopcodes ignorieren, falls es sich nicht um eine eigene Metadatei handelte (die VDI-Bindings sind in Kapitel 2.5 angegeben).

Beispiel: Die Versionsnummer 1.32 (GEM-Format = 100 * Hauptversionsnummer + Nebenversionsnummer) und der Name der Applikation (z.B. TEST) soll in eine Metadatei geschrieben werden.

```
WORD version;
BYTE app_name [80];

version = 132;
strcpy (app_name, "TEST");

intin [0] = 101; /* Subopcode für Version und Appname */
intin [1] = version;
for (i = 0; i < strlen (app_name); i++) int_in [i+2] = app_name [i];
v_write_meta (vdi_handle, strlen (app_name) + 2, intin, 0, ptsin);
```

Die Länge des intin-Feldes ergibt sich aus der Länge der Zeichenkette app_name plus 2 (Subopcode 101, Versionsnummer 1.32).

Die Funktion vm_filename wird dazu benutzt, den Namen einer Metadatei festzulegen. Wird eine Arbeitsstation als Metadatei geöffnet, so benutzt der Metadatei-Treiber standardmäßig die Datei GEMFILE.GEM, wenn Ausgaben auf diese Arbeitsstation getätigt werden. Da man aber meistens in eine Datei mit einem anderen Namen schreiben möchte, kann dieser umdefiniert werden. Man sollte diese Funktion sofort nach dem Öffnen der Arbeitsstation aufrufen (siehe auch VDITEST).

Beispiel: Es soll in die Datei „TEST.GEM“ geschrieben werden. Der Aufruf heißt

```
vm_filename (vdi_handle, "TEST");
```

Die beiden Funktionen vm_pagesize bzw. vm_coords sind erst ab der GEM-Version 2.X direkt verfügbar. Sie werden dazu benutzt, in einer Metadatei die Seitengröße bzw. das Koordinatensystem für diese Größe einzustellen. Aus dem Koordinatensystem und der Seitengröße ergibt sich dann die Pixelgröße der Punkte in einer Metadatei.

Die beiden Funktionen werden auf einem ATARI ST mit Hilfe des direkten Aufrufs des VDI nachgebildet und sind dann mit dem gleichen Namen verfügbar (siehe auch VDITEST).

Die Seitengröße wird in 1/10 mm angegeben. Die Breite und die Höhe einer Seite können getrennt eingestellt werden. Damit ist es auch möglich, Grafiken im Querformat zu erstellen (Breite größer als Höhe). Der C-Aufruf sieht wie folgt aus:

```
WORD pgwidth, pgheight;
vm_pagesize (vdi_handle, pgwidth, pgheight);
```

Bei der Angabe des Koordinatensystems müssen die x- und y-Werte der linken unteren und die x- und y-Werte der rechten oberen Ecke angegeben werden. Dabei muß beachtet werden, in welchem Koordinatentyp (NDC oder RC) man die Metadatei geöffnet hat. Entsprechend müssen die y-Werte verdreht werden (s.a. VDITEST). Der C-Aufruf sieht dann so aus:

```
WORD llx, lly, urx, ury;
vm_coords (vdi_handle, llx, lly, urx, ury);
```

```
llx: low left x
 :
ury: upper right y
```

Beispiel: Wir erstellen eine quadratische Grafik im RC-System mit einer Seitengröße von 5x5 cm. Pro Zentimeter sollen 10 Punkte gesetzt werden können (also millimetergenau), wobei auf der x- und y-Achse jeweils 50 Punkte (mit Koordinatenwerten 0 bis 49) gesetzt werden können. Danach soll eine horizontale Linie mit 3,5 cm Länge ausgegeben werden (z.B. von Punkt 0,0 nach Punkt 34,0).

```
WORD pgwidth, pgheight;
WORD llx, lly, urx, ury;
WORD pxy [4];

pwidth  = 500; /* 5 cm = 500/10 mm */
pheight = 500;

llx = 0;
lly = 49; /* 50 Punkte pro 5 cm = 1 Punkt/mm */
urx = 49;
ury = 0;

pxy [0] = 0;  /* x1 */
pxy [1] = 0;  /* y1 */
pxy [2] = 34; /* x2 */
pxy [3] = 0;  /* y2 */
```

```
vdi_handle = open_work (METAFILE, RC);
vm_pagesize (vdi_handle, pgwidth, pgheight);
vm_coords (vdi_handle, llx, lly, urx, ury);
v_pline (vdi_handle, 2, pxy);
close_work (METAFILE);
```

Man beachte, daß die Eckpunkte des Koordinatensystems im Wert immer um 1 vermindert sind, da sich die Anzahl der Punkte einer Achse (z.B. x-Achse) zu urx − llx + 1 berechnen. Ebenso berechnet sich die Länge einer horizontalen Linie zu x2 − x1 + 1, was in obigem Beispiel einen Wert von 35 ergibt. Ein Pixel in oben erzeugter Metadatei entspricht 1 mm Breite bzw Höhe.

- V_BIT_IMAGE
- V_XBIT_IMGAGE

Bit-Image-Dateien sind Dateien mit Suffix IMG, die komprimierte Pixelgrafiken enthalten. Der Algorithmus mag nicht das kompakteste Format erzeugen, aber genügt, um Pixelgrafiken platzsparend zu speichern. Deshalb sollte jedes Programm, welches Pixelgrafiken erzeugen kann, mindestens das GEM Bit-Image-Format einlesen und speichern können. Nur dann können alle Grafikpakete Daten miteinander austauschen.

Ein weiterer Vorteil beim Benutzen von Bit-Image-Dateien ist die Leichtigkeit, diese auf einem Drucker oder anderen Geräten (außer Bildschirm) auszugeben. Ein einziger VDI-Befehl (v_bit_image oder v_xbit_image) druckt die gesamte Grafik.

Die Funktion v_bit_image druckt die angegebene Datei auf das Ausgabegerät. Dabei kann das x-y-Verhältnis (aspect ratio) beibehalten, die Grafik horizontal und vertikal ausgerichtet sowie die Skalierung angegeben werden. Ein Rechteck, in das die Grafik passen muß, kann ebenfalls angegeben werden. Der C-Aufruf sieht wie folgt aus:

```
v_bit_image (vdi_handle, filename, aspect, x_scale, y_scale,
             h_align, v_align, xyarray);
```

Dabei haben die Variablen folgende Bedeutung:

Die Variable filename enthält den Dateinamen der IMG-Datei.

Aspect kann folgende Werte annehmen:
0 = Ignoriere x-y-Verhältnis
1 = Behalte x-y-Verhältnis bei
Wird das x-y-Verhältnis beibehalten, so erscheinen Kreise als Kreise, Quadrate als Quadrate usw. Das Bild wird auf dem Ausgabegerät dann entsprechend gedehnt. Dies kann unter Umständen schlecht aussehen, wenn z.B. feine Punktraster gedehnt oder gestaucht werden. Wird das Verhältnis ignoriert, so können aus Kreisen auch Ellipsen werden.

Die anderen Parameter beziehen sich auf das Rechteck, in das die Grafik hineinpassen soll.

Die Werte x_scale und y_scale geben an, ob eine Vergrößerung oder Verkleinerung auf den jeweiligen Achsen ganzzahlig oder in Brüchen durchgeführt werden soll. Dabei bedeuten:

0 = Skalierung mit Brüchen
1 = Ganzzahlige Skalierung

Bei einer Skalierung mit Brüchen paßt das Bild in das angegebene Rechteck in jedem Falle hinein und wird entsprechend gestaucht oder gedehnt (z.B. um den Faktor 2.5). Dies kann aber bei feinen Mustern schlecht aussehen, da man aus einem Pixel in der Grafik nicht 2.5 Pixel auf einem Drucker machen kann. Bei der Skalierung mit ganzen Zahlen kann dies nicht passieren. Dort werden z.B. aus einem Pixel 3 Druckerpixel, wenn eine Vergrößerung um den Faktor 3 stattfindet. Feine Muster erscheinen dann auch noch sauber.

Skaliert man ganzzahlig, so kann es sein, daß das Bild nur so skaliert werden kann, daß es im angegebenen Rechteck noch viel Platz gibt. Ist das Rechteck z.B. 2.5 mal so groß wie das Bild und skaliert man ganzzahlig, so wird nur um den Faktor 2 skaliert. Das Bild kann dann mittels der Variablen h_align und v_align in diesem Rechteck plaziert werden. Dabei können folgende Werte auftreten:

Ausrichtung horizontal:
h_align: 0 = links
1 = zentriert
2 = rechts

Ausrichtung vertikal:
v_align: 0 = oben
1 = mitte
2 = unten

Die Funktion v_xbit_image steht ab GEM/3 zur Verfügung. Damit ist es möglich, Bit-Image-Dateien rotiert ausgeben zu lassen. Außerdem kann die Hinter- und Vordergrundfarbe angegeben werden, wenn eine monochrome Bit-Image-Datei auf einem Farbdrucker ausgegeben werden soll. Der C-Aufruf sieht wie folgt aus:

```
v_xbit_image (vdi_handle, filename, aspect, x_scale, y_scale,
              h_align, v_align, rotate, background, foreground, xy);
```

Die Varible rotate gibt die Drehung der Grafik an. Meist ist nur ein Drehen in 90-Grad-Schritten möglich (vq_extnd gibt Auskunft darüber).

Die Variablen background sowie foreground enthalten den Farbindex (z.B 3 = Rot) des Hintergrundes bzw. des Vordergrundes, mit dem die Grafik ausgegeben werden soll, falls

es sich um eine einfarbige Grafik (Anzahl Farbebenen = 1) handelt.

Weitere Informationen über den Aufbau einer Bit-Image-Datei befinden sich in Kapitel 2.7.2.

– Hinweise über zukünftige Funktionen:

Ab GEM/3 Version 3.11 existieren weitere VDI-Funktionen, die sich mit Bezier-Funktionen und Postscript-Druckern beschäftigen. Bezier-Kurven und -Flächen können durch die Angabe von Stützstellen gezeichnet werden. Das Programm ARTLINE macht mit seinen Vektorfonts Gebrauch von diesen neuen Funktionen. Da sie aber offiziell im Programmer's Toolkit noch nicht dokumentiert sind, werden sie in einer späteren Auflage beschrieben. Für Interessierte befinden sich auf der Diskette das Bezier-Beispiel BEZIER.GEM. Es handelt sich um eine Bezier-Kurve. Man kann sich die Metadatei mit dem Programm DUMPGEM anschauen, um die neuen Funktionen zu erkennen. Man kann die Datei auch über das OUTPUT ausgeben, wobei die Bezier-Kurve natürlich nur mit Polygonzügen angenähert wird.

– Zusammenfassung VDI:

Wir haben erfahren, welche Rolle das VDI im GEM und im Betriebssystem eines Rechners spielt. Außerdem stellt das VDI eine große Menge grafischer Funktionen zur Verfügung, die von Linienzeichnen (v_polyline) bis zum kompletten Ausdruck von Pixelgrafiken (v_bit_image) reichen. Mit diesen Funktionen ist es möglich, Grafiken auf unterschiedlichen Geräten auszugeben, ohne daß dies für jedes Gerät neu programmiert werden muß. Dabei übernahmen die Treiber der Ausgabegeräte (workstations) die Funktion, jede über das VDI ankommende Grafik auf dem entsprechenden Gerät in optimaler Qualität auszugeben.

Wir lernten außerdem die Rolle der Datei ASSIGN.SYS bzw. der GEM/3-Funktion v_get_driver_info kennen, die Gerätenummern, Treiber und Zeichensätze der Geräte verbindet und wie die interne Kommunikation mit dem VDI abläuft (contrl, intin Felder je nach Funktionsart füllen).

Am Ende dieses Abschnitts wurden dann alle wichtigen VDI-Funktionen genau erläutert. Dort wurde gezeigt, wie man Zeichensätze lädt und die Indizes zum Auswählen der Sätze bekommt. Rasterfunktionen wurden genau erläutert, da das Benutzen dieser Funktionen die einzige hardwareunabhängige Möglichkeit darstellt, Bildschirminhalte zu scrollen bzw. zu kopieren. Das Format eines Grafikblockes spielte dort eine wichtige Rolle für die Behandlung von Icons (Umwandlung Standardformat <-> gerätespezifisches Format).

Schließlich wurde gezeigt, wie man Metadateien erstellt (VDITEST) und wir erfuhren, daß das Benutzen von Metadateien die einzige saubere Möglichkeit ist, Vektorgrafiken

(nur mit VDI-Befehlen erzeugt) zwischen verschiedenen Programmen auszutauschen. Das Austauschen dieser Grafiken ist nicht nur programmunabhängig, sondern auch rechnerunabhängig. Metadateien sind z.B. zwischen MS-DOS- und ATARI-Rechnern beliebig austauschbar.

## 2.4 AES

Nachdem wir im letzten Kapitel das VDI kennengelernt haben und nun wissen, wie man Grafik auf einem Bildschirm erzeugt, werden wir in diesem Kapitel auf die zweite wichtige Komponente des GEM eingehen. Es handelt sich um das AES (Application Environment Services), welches sich in 12 Funktionsgruppen aufteilt. Die Funktionen in diesen Gruppen dienen dazu, einem Anwendungsprogrammierer das Erzeugen von AES-Komponenten zur Verfügung zu stellen. Komponenten des AES sind Bildschirmfenster, Menüzeilen, Drop-Down-Menüs, Ereignissteuerung durch eine Maus, Formularverwaltung (sogenannte Dialogboxen) und andere Funktionen.

Jeder, der schon einmal mit einem GEM-Programm gearbeitet hat, weiß, was mit Dialogboxen, Fenstern und Menüs gemeint ist. Wir gehen davon aus, daß ein GEM-Entwickler mindestens ein GEM-Programm (z.B. den DESKTOP) gut kennt und bedienen kann. Daher ersparen wir uns langwierige und langweilige Einführungen und konzentrieren uns auf die wesentlichen und interessanten Aspekte der GEM-AES-Programmierung. Auch werden wir nur auf solche AES-Funktionen genauer eingehen, die in anderen Büchern gar nicht oder falsch erläutert wurden und solche, die in neueren GEM-Versionen hinzugekommen sind.

Grundsätzlich gibt es 2 Arten von GEM-Programmen: Applikationen und Desk-Accessories. Der Unterschied ist aber nur gering. Applikationen werden per Mausklick vom GEM-Desktop aus gestartet und laufen dann im Vordergrund als Hauptapplikation ab. Desk-Accessories werden beim Start des GEM-Systems geladen, bleiben resident im Speicher und laufen im Hintergrund quasi parallel ab. Beispiele sind: Uhr, Druckerspooler, Kontrollfeld usw. Sie können von jedem GEM-Programm aus aktiviert werden. Die Programmierung und Handhabung ist aber bei beiden Programmtypen gleich. Im Beispielprogramm in Kapitel 6 wird gezeigt, wie ein Programm gleichzeitig als Applikation oder als Accessory benutzt werden kann.

### 2.4.1 Speicherbelegung

Das AES setzt auf dem VDI und dem Betriebssystem auf und kann daher nicht ohne VDI benutzt werden (im Gegensatz dazu kann das VDI ohne das AES gestartet werden). Wird beispielsweise ein Fenster gezeichnet, so werden Linien und Füllmuster, sowie Textausgaben über das VDI gemacht.

Die folgende Abbildung soll verdeutlichen, wie das GEM-System im Speicher liegt, wenn eine GEM-Applikation gestartet wird.

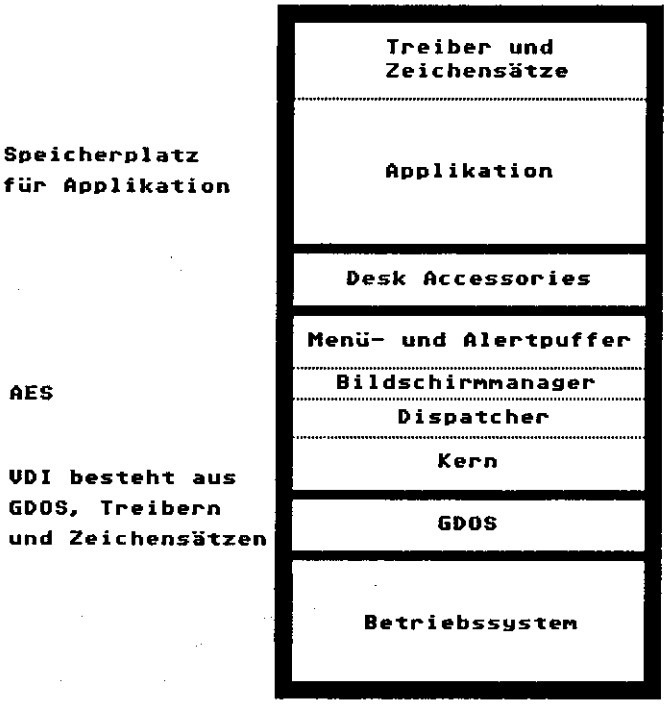



**Abb. 2.10:** Speicheraufteilung einer GEM-Applikation

Wie man sieht, liegt das GEM-System (GDOS, Kernel,...,Alertpuffer) ständig im Speicher. Der Menü- und Alertpuffer wird dazu benötigt, den Bildschirmhintergrund zu retten, wenn ein Menü herunterklappt bzw. wenn eine Alertbox erscheint.

Darüber liegen alle geladenen Desk-Accessories. Der Rest des Speichers bleibt frei für GEM-Applikationen. Eine GEM-Applikation kann mittels v_opnwk bzw. vst_load _fonts Treiber bzw. Zeichensätze nachladen, die dann oberhalb der Applikation im Speicher Platz finden.

### 2.4.2 AES Komponenten

Das AES besteht aus einem Kern, der die AES-Funktionen bereitstellt, dem Bildschirmmanager, einem Dispatcher sowie einem Menü- und einem Alert-Puffer. Alle AES-Funktionen werden sofort beim Aufruf ausgeführt. Ausnahmen bilden die Funktionen aus der Ereignis-Bibliothek (Events). Ein Aufruf solch einer Funktion veranlaßt den Dispatcher auf einen anderen Prozeß umzuschalten. Prozesse in einem GEM-System sind:

a) die Hauptapplikation
b) eines der Desk-Accessories (bis zu 6 Stück)
c) der Bildschirmmanager

Diese Prozesse laufen alle parallel ab, so daß auf einem Einprozessor-System ständig zwischen ihnen hin- und hergeschaltet werden muß.

Der Bildschirmmanager ist ein Hintergrund-Prozeß, der die Bildschirmverwaltung sowie Tastatur- und Mauseingaben steuert und immer dann aktiv wird, wenn die Maus folgende Bereiche betritt:

a) die Menüzeile und Drop-Down-Menüs
b) Titel- und Informationszeile eines Fensters
c) Randkomponenten eines Fenster (z.B. Schieber)

Da GEM bisher nur auf Einprozessor-Systemen implementiert wurde, kann es kein echtes Multitasking geben, d.h. der Dispatcher hat keine Möglichkeit, zwischen o.g. Prozessen umzuschalten. Dies kann er nur dann, wenn Programme AES-Aufrufe tätigen. Hängt z.B. ein Programm in einer Endlosschleife, in der kein AES-Aufruf erfolgt, so hängt das ganze System. Deshalb sollten in Programmschleifen, in denen nur gerechnet wird, ab und zu AES-Aufrufe getätigt werden. Die Funktion appl_yield gibt dem Dispatcher die Möglichkeit auf einen anderen Prozeß umzuschalten. Andere Prozesse, z.B. ein Drucker-Spooler als Desk-Accessory, können dann noch aufgerufen werden bzw. können weiterlaufen, obwohl das Hauptprogramm hängt. Außerdem wird die Zeit zwischen den Prozessen gerechter verteilt, wenn jeder Prozeß hin und wieder seine Aktivität durch den appl_yield-Aufruf hinten anstellt.

### 2.4.3 AES-Funktionsgruppen

Die o.g. Funktionsgruppen sind in einzelnen Bibliotheken (Libraries) abgelegt. Der Name der Bibliothek wird in Englisch gehalten, damit man auch mit den Originalentwicklungsunterlagen von Digital Research keine Probleme hat, wenn man Funktionen nachschlagen muß. Der Inhalt der Bibliotheken sind:

| Bibliothek | Kurzbeschreibung |
|---|---|
| Application | Initialisierung und Kommunikation |
| Event | Eingabe- und Ereignisbehandlung |
| Menu | Verwaltung der Menüzeile |
| Object | Verwaltung von Dialogboxen |
| Form | Eingabeverwaltung von Dialogboxen |
| Graphics | Rechteckverwaltung |
| Scrap | Verwaltung für Datenaustausch |
| File Selector | Dateiauswahl |
| Window | Fensterverwaltung |

| | |
|---|---|
| Resource | Resourceverwaltung |
| Shell | Verwaltung von Informationen der Shell |
| Extended Graphics | Weitere Rechteckfunktionen |

Um einen besseren Überblick über die Funktionen des AES zu bekommen, sollen die einzelnen Funktionsgruppen kurz erläutert werden.

– Application-Library

In dieser Bibliothek sind Funktionen enthalten, um eine GEM-Applikation zu initialisieren und zu terminieren. Aber auch die Kommunikation mit anderen Programmen und Accessories kann damit vorgenommen werden (Senden und Empfangen von Nachrichten). Ab GEM 2.X kann außerdem die Konfiguration der angeschlossenen Laufwerke bestimmt werden.

– Event-Library

GEM-Programme sind ereignisgesteuert, d.h. ein Programm wartet an einer bestimmten Stelle auf das Eintreten von Ereignissen, die dann abgehandelt werden. Ereignisse können Mausklicks, Tastendrucke, Mausbewegungen, Zeitintervalle oder auch Nachrichten (vom GEM System oder anderen Applikationen) sein. Für die Abhandlung von Ereignissen stehen die Funktionen der Ereignis-Bibliothek zur Verfügung.

– Menu-Library

Menüzeilen in GEM-Programmen erscheinen am oberen Bildschirmrand. Die Menü-Bibliothek stellt Funktionen zur Verfügung, um Menüzeilen anzuzeigen, einzelne Menütitel grau (schwach) darzustellen und Häkchen (check markers) vor einen Menütitel zu stellen. Außerdem können in das Accessory-Menü Einträge aufgenommen werden, sowie Texte von Menüeinträgen verändert werden.

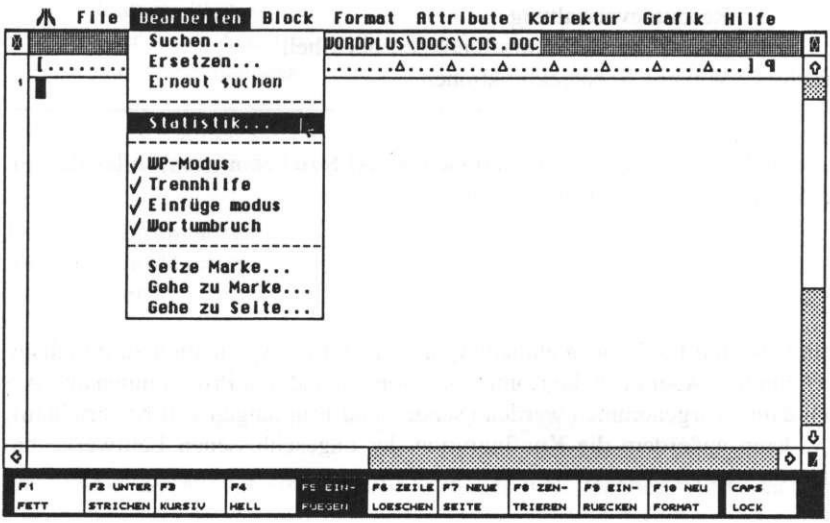



**Abb. 2.11:** Menüzeile eines GEM-Programmes

– Object-Library

Die Funktionen der Objekt-Bücherei sind wichtig für das Verwalten von Objektbäumen (Dialogboxen). Ein einziger Befehl zeichnet eine komplette Dialogbox oder findet ein Objekt einer Dialogbox, welches sich unter dem Mauszeiger befindet.

– Form-Library

Zusammen mit der Objekt-Bücherei übernimmt die Formular-Bücherei weitere Verwaltungsarbeit für Dialogboxen. So kann mit einem einzigen Befehl die komplette Abarbeitung eines Dialoges gemacht werden. Aber auch Fehlermeldungen können elegant ausgegeben werden.

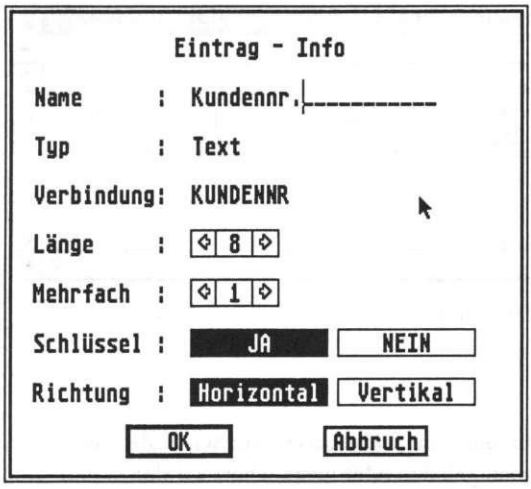



**Abb. 2.12:** GEM-Dialogbox (Formular)

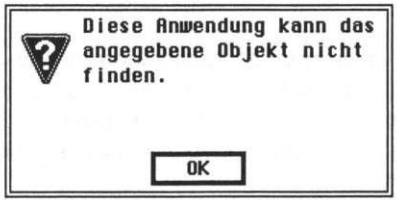



**Abb. 2.13:** GEM-Fehlermeldung

– Graphics-Library

In der Grafik-Bibliothek findet man Funktionen für die Verwaltung von Rechtecken. So kann ein Rechteck gezogen werden, das sich mit dem Mauszeiger automatisch mitbewegt, oder Rahmen können mit der Maus erstellt werden (z.B. das Einrahmen von Dateien auf dem Desktop). Die Mausform kann verändert oder gänzlich abgeschaltet werden.

– Scrap-Library

Die Scrap-Bibliothek wurde in den meisten GEM-Programmen sehr stiefmütterlich behandelt. Mit Hilfe dieser Funktionen ist es möglich, den Datenaustausch zwischen Programmen zu standardisieren. Deshalb wird in Kapitel 2.4.4 genau darauf eingegangen. Das Beispielprogramm aus Kapitel 6 ist eine komplette Implementierung eines Klemmbrettes, welches auch als Accessory verwendet werden kann.

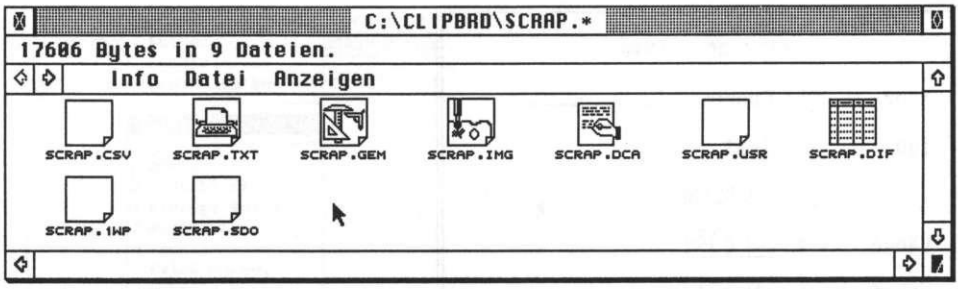



**Abb. 2.14:** Inhaltsverzeichnis des GEM-Klemmbrettes

– File-Selector-Library

Die Dateiauswahl-Bibliothek besteht nur aus einer Funktion. Sie wird aber in jedem GEM-Programm benutzt, bei denen Dateien geladen oder gespeichert werden sollen. Der Aufruf dieser Funktion stellt eine Dialogbox zur Verfügung, in der der Benutzer die Inhaltsverzeichnisse aller angeschlossenen Laufwerke bzw. Partitionen der Festplatte nach Dateinamen durchforsten kann.

– Window-Library

Die Fenster-Bibliothek stellt die wichtigsten Funktionen einer GEM-Applikation zur Verfügung. Mit ihnen können Fenster kreiert, geöffnet und geschlossen werden. Auch die Randkomponenten eines Fensters können bestimmt werden, also ob ein Fenster sog. Schieber hat oder ob es vergrößert/verkleinert werden darf. Auch der Name des Fensters kann gesetzt werden.

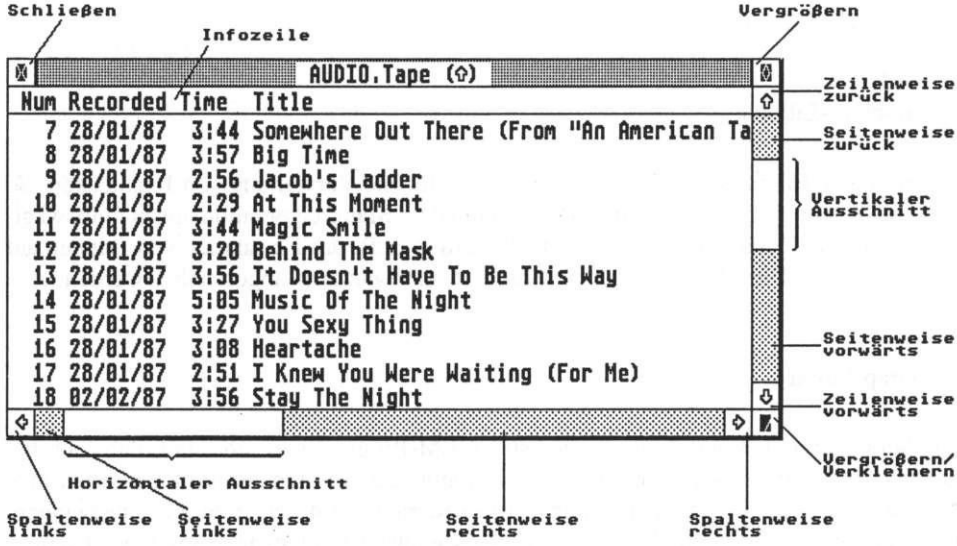



**Abb. 2.15:** GEM-Fenster mit Komponenten

– Resource-Library

Unter Resourcen einer GEM-Applikation werden alle Teile des Programmes verstanden, die geändert werden müssen, wenn das Programm z.B. in eine andere Sprache übersetzt werden soll. GEM-Applikationen werden also üblicherweise getrennt in den reinen Programmteil und den Resourceteil. In den Resource-Dateien befinden sich Menüleisten und Drop-Down-Menüs, Dialogboxen und Fehlermeldungen, Piktogramme oder einfach freie Texte. Mit den Funktionen der Resource-Bibliothek können die Resourcen geladen werden und die Adressen der einzelnen Objekte zum späteren Gebrauch geholt werden. Resource-Dateien enden mit dem Suffix RSC und können mit dem sogenannten Resource-Construction-Set erzeugt werden.

– Shell-Library

Die Shell-Bibliothek wird dazu benutzt, Informationen aus der Programmumgebung zu lesen oder solche Informationen zu schreiben.
Auch das Aufrufen von anderen Programmen aus einem GEM-Programm heraus ist möglich. Welches Programm beim Verlassen eines aufgerufenen Programmes gestartet werden soll, kann ebenfalls bestimmt werden. Im Normalfall wird es der GEM-Desktop sein.

– Extended-Graphics-Library

Die Funktionen aus der erweiterten Grafik-Bibliothek sind ab GEM 2.X verfügbar und ersetzen die Funktionen graf_growbox und graf_shrinkbox. Mit Hilfe der beiden Funktionen ist es möglich, ein sich ausdehnendes oder ein schrumpfendes Rechteck zu zeichnen.

### 2.4.4 Kommunikation mit dem AES

Alle GEM-Programme nutzen das AES, das alle nötigen Funktionen für die Erstellung von Anwendungsprogrammen bietet, um eine moderne Benutzerschnittstelle zu schaffen. Für Programme, die in einer höheren Programmiersprache geschrieben sind, erfolgt die Nutzung des AES einfach durch den Aufruf der Funktionsnamen.

Assembler-Programmierer haben es da schwerer. Sie müssen verschiedene Strukturen beachten, um die Parameter an das AES zu liefern. Obwohl es kaum Sinn macht, AES-Befehle in Assembler zu nutzen, soll die Parameterübergabe doch erklärt werden.

Das AES wird über ein einziges Unterprogramm aufgerufen, dem 6 Parameter übergeben werden. Es handelt sich um 6 Felder (arrays), wobei der Grundtyp eines Feldes aus einem Rechnerwort oder Langwort, also 16 oder 32 Bit bestehen muß. Die Felder werden wie folgt benannt:

- control (Funktionsangabe)
- global (globale Informationen wie z.B. GEM Version)
- int_in (ganzzahlige Eingaben)
- int_out (ganzzahlige Ausgaben)
- addr_in (Eingabe-Adressen, Grundtyp LONG)
- addr_out (Ausgabe-Adressen, Grundtyp LONG)

```
EXTERN WORD control [];
EXTERN WORD global [];
EXTERN WORD int_in [];
EXTERN WORD int_out [];
EXTERN LONG adr_in [];
EXTERN LONG adr_out [];
```

EXTERN bedeutet in diesem Zusammenhang, daß die Felder beim Compilerlauf vom Programmierer benutzt werden können (definiert sind sie in den AES-Bindings). WORD bedeutet 16-Bit-Werte, LONG 32-Bit-Werte. Die Bedeutung solcher Portabilitäts-Makros wird in Kapitel 3 erklärt.

Die o.g. Felder werden beim Aufruf der AES-Funktionen von höheren Programmiersprachen aus automatisch von den sogenannten AES-Bindings gefüllt. Die Felder enthalten folgende Informationen:

a) control

```
control [0] = Funktionsnummer
control [1] = Anzahl der Elemente im Feld int_in
control [2] = Anzahl der Elemente im Feld int_out
control [3] = Anzahl der Elemente im Feld adr_in
control [4] = Anzahl der Elemente im Feld adr_out
```

b) global

Dieses Feld wird beim Aufruf der Funktion appl_init automatisch gefüllt.

```
global [ 0] = GEM Versionsnummer
global [ 1] = Vordergrundapplikationen, die gleichzeitig ablaufen können.
global [ 2] = eindeutiger Bezeichner für die Applikation (gl_apid)
global [ 3] - global [4] = ob_spec von Fenster 0
global [ 5] - global [6] = Zeiger auf Basisbäume der geladenen Resource-Datei
global [ 7] - global [8] = Zeiger auf Adresse, an der die Resource-Datei geladen
                           wurde
global [ 9] = Länge der zuletzt geladenen Resource-Datei
global [10] = Anzahl der Farbebenen (1 - 4)
global [11] = reserviert
```

global [12] = reserviert
global [13] = Bit-Vektor, der angibt, welche Laufwerke auf dem GEM-Desktop installiert wurden. Ist das Bit gesetzt, so wurde das Laufwerk installiert. Dabei gelten folgende Bitbelegungen:
Bit 15 = Laufwerk A
Bit 14 = Laufwerk B
Bit 13 = Laufwerk C
usw.
Bit 0 = Laufwerk P
global [14] = Bit-Vektor, der angibt, ob das installierte Laufwerk ein Diskettenlaufwerk oder eine Harddisk ist. Ein gesetztes Bit bedeutet, daß es sich um eine Harddisk handelt.

– Der Parameter-Block

Er hat nur Bedeutung, wenn man selbst die Parameter an das AES übergeben möchte. Im Normalfall werden über High-Level-Funktionsaufrufe (C, Pascal usw.) die benötigten Parameter aufbereitet und an das AES übergeben.

Um die Parameter selbst an das AES zu übergeben, muß der sogenannte Parameter-Block erstellt werden. Er hat folgenden einfachen Aufbau (C-Schreibweise):

```
#define C_SIZE  5
#define G_SIZE 15
#define I_SIZE 16
#define O_SIZE  7
#define AI_SIZE 2
#define AO_SIZE 5

typedef struct gemblkstr
{
  LONG gb_pcontrol;
  LONG gb_pglobal;
  LONG gb_pintin;
  LONG gb_pintout;
  LONG gb_padrin;
  LONG gb_padrout;
} GEMBLK;

EXTERN WORD   gem ();

GLOBAL GEMBLK gb;
GLOBAL UWORD  control [C_SIZE];
GLOBAL UWORD  global [G_SIZE];
GLOBAL UWORD  int_in [I_SIZE];
```

```
GLOBAL UWORD  int_out [O_SIZE];
GLOBAL LONG   addr_in [AI_SIZE];
GLOBAL LONG   addr_out [AO_SIZE];
GLOBAL LONG   ad_g;

WORD gem_if (opcode)
WORD opcode;
{
  WORD i;
  BYTE *pctrl;

  control[0] = opcode;

  pctrl = &ctrl_cnts[(opcode - 10) * 3];
  for(i=1; i<=CTRL_CNT; i++) control[i] = *pctrl++;

  gem (ad_g);
  return ((WORD)RET_CODE);
}



struct aes_parameters
{
  WORD *control;
  WORD *global;
  WORD *int_in;
  WORD *int_out;
  LONG *addr_in;
  LONG *addr_out;
};

struct aes_parameters pblock =
  {control, global, int_in, int_out, addr_in, addr_out};
```

Die Variable pblock enthält also die Adressen der 6 AES-Parameter. Nun muß nur noch die Adresse der Variablen pblock sowie eine ID-Nummer in die entsprechenden Register geladen werden. Anschließend erfolgt ein Software-Interrupt oder ein Trap-Befehl.

a) INTEL-Modell (IBM PC's und Kompatible)

Die Adresse des Parameter-Blockes, welcher sich auf dem Stack befindet, muß in die Register AX:BX, sowie der ID-Wert 200 (Hex $C8) in das Register CX übertragen werden. Anschließend muß der Interrupt $EF ausgeführt werden.

In 8086-Assembler könnte dies wie folgt aussehen:

```
_gem:   push    bp
        mov     bp,sp
        mov     cx,200
        mov     dx,0
        mov     bx,4[bp]
        mov     ax,6[bp]
        mov     es,ax
        int     0EFh
        pop     bp
        ret
```

b) 68000-Modell (ATARI ST, TT)

Die Adresse des Parameter-Blockes, welcher sich auf dem Stack befindet, muß in das Register D1.L, sowie der ID-Wert 200 (Hex $C8) in das Register D0.W übertragen werden. Anschließend muß der Interrupt TRAP #2 ausgeführt werden.

In 68000-Assembler könnte dies wie folgt aussehen:

```
_gem:   move.l  4(A7),D1
        move.w  #200,D0
        trap    #2
        rts
```

Für beide Modelle gilt dann, daß das AES angesprungen wird und die Parameter ausgewertet werden. Das AES füllt die Ausgabeparameter int_out und addr_out, da es ja die Adressen der beiden Felder kennt. Von dort kann dann die Applikation die Werte wieder übernehmen. Benutzt man eine höhere Programmiersprache, übernehmen die Bindings (AES-Libraries) die Übertragung der Werte aus den Ausgabearrays in die Parameter der Funktionsaufrufe.

### 2.4.5 Allgemeine Aufruffolgen im AES

An dieser Stelle möchten wir uns ansehen, welche AES-Funktionen zu welchem Zeitpunkt benutzt werden müssen, um eine Standardapplikation aufzubauen. Eine Standardapplikation wie der GEM-Desktop, GEMDRAW oder WORDPLUS kann man sich wie folgt vorstellen:

Das Programm wird initialisiert, besitzt eine Menüleiste am oberen Bildschirmrand, benutzt Piktogramme, die auf dem Desktop abgelegt sind, zeigt Dialogboxen, wenn bestimmte Menüpunkte angewählt werden und es können Bildschirmfenster kreiert, geöffnet, geschlossen und verschoben werden. Dabei muß das Innere des Fensters zu bestimmten Zeitpunkten auf den neuesten Stand gebracht werden.

a) Initialisierung der Applikation

Bevor das Hauptprogramm gestartet wird, wird vom Initialisierungcode des Compilers der Speicher freigegeben, der nicht von der Applikation benötigt wird. Dieser Initialisierungscode (meist Startupcode genannt) wird beim Linkvorgang der Applikation vorangestellt und übernimmt diese Aufgabe. Man braucht sich selbst nicht darum zu kümmern.

Die AES-Datenstrukturen (Parameterblock und entsprechende Felder) müssen deklariert und initialisiert werden. Auch dies wird von den sogenannten Bindings schon erledigt, wenn man in höheren Programmiersprachen seine Applikation erstellt.

Der erste Aufruf, den eine Applikation machen muß, ist dann appl_init. Die Funktion gibt die Applikations-ID (ap_id) zurück, die im Feld global (s.o.) abgelegt wird. Daran kann das GEM-System jedesmal die Applikation erkennen, wenn weitere AES-Aufrufe getätigt werden.

b) Bildschirmauflösung bestimmen

Alle professionellen GEM-Programme laufen mit beliebigen Bildschirmauflösungen. Ihnen ist es egal, ob der Bildschirm 640x200 oder 1280x960 Pixel breit bzw. hoch ist. Alle anderen Programme, die dies nicht können, sind keine korrekten GEM-Programme und ignorieren die Idee, die GEM zu verwirklichen versucht: Totale Hardwareunabhängigkeit aller Software!

Glücklicherweise existieren heute so viele Grafikkarten und Bildschirmgeräte mit verschiedenen Auflösungen und Farbebenen, daß niemand mehr daran vorbeikommt, saubere GEM-Applikationen zu schreiben.

Um die Bildschirmauflösung zu bestimmen, müssen VDI-Aufrufe getätigt werden. Da der Bildschirm bei der Initialisierung des GEM schon durch das VDI physikalisch geöffnet wurde (v_opnwk), kann der Bildschirm nur noch virtuell geöffnet werden (v_opnvwk). Dazu muß das VDI-Handle des physikalischen Gerätes Bildschirm bekannt sein. Dies erhält man durch die AES-Funktion graf_handle. Damit öffnet man den Bildschirm virtuell und bekommt im work_out-Feld sämtliche Informationen über das Gerät Bildschirm.

c) Laden der Resource-Datei

Jetzt kann entsprechend der Bildschirmauflösung die Resource-Datei geladen werden. Einige Programme benutzen verschiedene Dateien je nach Bildschirmauflösung. Dies läßt sich leider nicht immer vermeiden, da z.B. Piktogramme, die für eine hohe Auflösung erstellt wurden, in einer niederen Auflösung verzerrt erscheinen (z.B. doppelt so hoch) oder nicht ganz in den Rahmen passen, für den sie vorgesehen sind.

Beim Laden der Resource-Datei mit rsrc_load werden vom GEM noch einige Umwandlungen vorgenommen. Alle Zeiger, die sich in einer Resource-Datei befinden, sind relativ zu 0 bzw. relativ zu einem Feld (siehe auch Kapitel 5.3 im Buch „Softwareentwicklung auf dem ATARI ST", Hüthig Verlag). Sie werden entsprechend der Ladeadresse vom Resource-Lader angepaßt.

Die X- und Y-Werte sowie die Breite und Höhe von Objekten in einer Resource-Datei sind in Zeichenkoordinaten angegeben. Dabei geht das GEM-System davon aus, daß man einen Standardbildschirm von 80 Zeichen zu 25 Zeilen benutzt. Ein Objekt mit der Breite 4 und der Höhe 2 ist also so breit wie 4 Zeichen und so hoch wie 2 Zeichen des Standardzeichensatzes (vst_font mit Parameter 1). Beim Laden der Resource-Datei werden also X, Y, Breite und Höhe jedes Objektes mit der Größe des Standardzeichensatzes multipliziert. Danach stehen die Werte in Pixelkoordinaten des RC-Systems im Objektbaum. Dann können sie von der Appliaktion aus weiter benutzt werden (siehe auch Funktion rsrc_obfix).

d) Holen der Resource-Adressen

Nachdem die Resource-Datei erfolgreich geladen wurde, muß man sich zu jedem Objektbaum (Menüzeile, Dialogbox, Fehlermeldung etc.) die Adresse besorgen. Diese speichert man in einer Variablen vom Typ Zeiger auf OBJECT ab. Die Adresse eines Objektbaumes kann man sich mit der Funktion rsrc_gaddr besorgen. Man ruft die Funktion also für jeden Baum auf. Die Namen (C-Konstanten), die bei diesem Aufruf angegeben werden, können mit dem Resource-Construction-Set vergeben werden.

e) Anzeigen der Menüzeile

Die Menüzeile ist ein Objektbaum, deren Adresse man sich gerade besorgt hat. Durch den Aufruf menu_bar, bei dem man die Adresse und den Wert TRUE übergibt, wird die Menüzeile vom AES automatisch gezeichnet.

f) Piktogramme auf dem Desktop darstellen

Der Desktop ist der Bereich des Bildschirms unterhalb der Menüzeile. Dieser Bereich wird auch als Desktop-Fenster bezeichnet, dessen Kennung mit 0 bei Aufrufen von Funktionen aus der Fenster-Bibliothek angegeben wird.

Grundsätzlich gibt es 2 Möglichkeiten, Piktogramme auf dem Desktop-Fenster darzustellen: Man zeichnet die Piktogramme selbst oder man installiert ein neues Desktop-Fenster.

Bevor man eine der beiden Möglichkeiten wählt, müssen jedoch die Piktogramme umgewandelt werden. In der Resource-Datei liegen sie im Standardformat vor, und sie müssen in das gerätespezifische Format gewandelt werden (siehe auch VDI-Funktion vr_trnfm).

Das Desktop-Fenster hat eine bestimmte Größe (Bildschirmgröße abzüglich Menüleiste). Diese Größe sollte jetzt bestimmt werden. Dazu bedient man sich der Funktion wind_get mit Fensterkennung 0 und der Konstanten WF_WXYWH. Man erhält die Werte X, Y, Breite und Höhe des Desktop-Fensters auf dem gesamten Bildschirm. Bei einer Auflösung von 640x400 werden z.B. die Werte

```
x = 0
y = 19
width = 640
height = 381
```

zurückgegeben, je nachdem, wie hoch die Menüleiste ist, was wiederum vom verwendeten Zeichensatz abhängt.

Möchte man die Piktogramme selbst zeichnen, so kann dies jetzt mit der Funktion objc_draw gemacht werden. Werden diese Piktogramme irgendwann aber einmal verdeckt (z.B. durch ein Fenster eines Accessories) und kommen wieder zum Vorschein, so muß man sich selbst darum kümmern, sie neu zu zeichnen.

Aus diesem Grund raten wir zur zweiten Methode. In diesem Falle installieren wir das Desktop-Fenster mit der Kennung 0 neu und übergeben einen Objektbaum, der die Piktogramme für den neuen Desktop enthält. Ist der neue Desktop erst einmal installiert (siehe wind_set mit der Konstanten WF_NEWDESK), dann übernimmt das AES die Update-Arbeit, wenn Objekte z.B. durch Verschieben eines Fensters wieder zum Vorschein kommen.

g) Warten auf Benutzerereignisse

Nachdem der Bildschirm (Menüzeile und Desktop) der Applikation aufgebaut ist, sollte man auf Ereignisse eines Benutzers warten. Solche Ereignisse können sein:

- Drücken einer Taste
- Drücken eines Mausknopfes
- Mausbewegungen
- Nachrichten von AES-Prozessen (z.B. vom Fenstermanager)
- Ablauf einer bestimmten Zeitdauer

Man kann auf jedes Einzelereignis warten. D.h. wartet man auf einen Mausknopf, so kann man nicht gleichzeitig auf das Drücken einer Taste warten. Zu diesem Zweck gibt es ein kombiniertes „Warten" (event_multi), das auf beliebig kombinierbare Ereignisse warten kann.

Tritt das Ereignis auf, so bekommt man eine Meldung des GEM-Systems in den Nachrichtenpuffer gesetzt, den man auslesen muß. Nachrichten können sein: Accessory wurde geöffnet, Fenster wurde verschoben oder geschlossen uvm. Man unternimmt dann die entsprechende Aktion (siehe auch EVENT.C) und wartet auf die nächste Nachricht.

h) Menüauswahl

Die Menüzeile wird vom Bildschirmmanager automatisch verwaltet, d.h. das Herunterklappen und Invertieren von Menüeinträgen muß nicht selbst vorgenommen werden. Am Beispiel des SCRAP-Programmes, das sich auf der Diskette befindet, wollen wir uns in etwa den Ablauf verdeutlichen:

- Die Applikation wartet auf ein Ereignis, meist ein Multi-Event.
- Der Benutzer bewegt die Maus in das SCRAP-Menü, wobei der Bildschirmmanager das Menü herunterklappt, den Hintergrund vorher rettet und die Menüeinträge entsprechend den Mausbewegungen invertiert.
- Der Benutzer klickt in den Menüpunkt „Über SCRAP..."
- Der Bildschirmmanager schreibt die Nachricht MN_SELECTED (Menü ausgewählt) in den Nachrichtenpuffer der Applikation und gibt die Nummer im Objektbaum „Menüleiste" mit zurück. Diese Nummer kann über Konstanten abgefragt werden. Die Konstanten werden im Resource-Construction-Set vergeben.
- Das Warten der Applikation auf ein Ereignis wird beendet.
- Die Applikation interpretiert das Ereignis und den Nachrichtenpuffer und bringt eine Dialogbox auf den Schirm.
- Der Menütitel „SCRAP" bleibt noch so lange invers, bis der Benutzer in der Dialogbox „OK" angewählt hat. Jetzt setzt die Applikation den Menütitel durch die Funktion menu_tnormal wieder auf Normalschrift.

i) Dialogbox anzeigen und abhandeln

Um die oben beschriebene Dialogbox anzuzeigen, werden folgende Schritte unternommen:

- Die Adresse des Menübaumes ist bekannt (init_resource in RESOURCE.C).
- Durch die Funktion form_center wird erreicht, daß die Dialogbox in der Mitte des Bildschirms erscheinen wird.
- Die Funktion form_dial mit der Konstanten FMD_START bewirkt, daß der Bereich, den die Dialogbox bedeckt, als belegt gekennzeichnet wird. Der Bildschirmmanager wird nach Beenden des Dialoges diesen Bereich allen Applikationen anbieten, damit sie das Innere eines Fensters restaurieren können. Die Randteile von Fenstern sowie der Desktop selbst wird automatisch restauriert.
- Jetzt kann mit der Funktion objc_draw die gesamte Dialogbox gezeichnet werden.
- Durch Aufruf der Funktion form_do wird der gesamte Dialog mit der Dialogbox vom AES abgehandelt.
- Hat der Benutzer auf „OK" geklickt, so wird der Knopf invers dargestellt und der Dialog ist beendet. Jetzt kann die Applikation nachschauen, welche Werte sich im Objektbaum geändert haben und entsprechend reagieren. Der Status (ob_state) des „OK"-Knopfes hat jetzt den Wert „SELECTED", so daß die Applikation erkennen kann, das dieses Objekt aus dem Objektbaum gewählt wurde. Mit der Funktion undo_state kann dieser Status wieder geändert werden, sonst käme der „OK"-Knopf beim nächsten Aufruf schon invertiert hoch.

- Jetzt kann die AES-Funktion form_dial mit der Konstanten FMD_FINISH aufgerufen werden. Der Bildschirmmanager sendet jetzt an alle Applikationen die Meldung, daß der Bildschirmbereich wieder frei wurde, damit sie ihren Teil des Bildschirms restaurieren können.

j) Menüauswahl über Tastatur

Durch die Angabe von MU_KEYBD im Multi-Event wartet die Applikation zusätzlich auf Tastendrücke. Wurde eine Taste gedrückt, sollte man den entsprechenden Menütitel invertieren (menu_tnormal), damit der Benutzer erkennt, daß eine Menüauswahl stattgefunden hat. Nach Beendigung der Aktion muß der Menütitel wieder auf Normalschrift zurückgesetzt werden.

k) Selektieren eines Piktogrammes

Wenn der Benutzer ein Piktogramm einmal mit der Maus angeklickt hat, muß folgendes passieren:

- Beim Warten mittels dem event_multi-Befehl wurde angegeben, daß ein Mausklick erwartet wird (MU_BUTTON). Das Warten auf das Ereignis wird beendet, wobei x- und y-Koordinaten der Maus zurückgegeben werden.
- Mit dem AES-Befehl objc_find und der Angabe der Mauskoordinaten sowie des Objektbaumes (z.B. der neue Desktop) bestimmt das AES die Objektnummer, also das gewählte Piktogramm. Der Wert ob_state des gewählten Objekts muß jetzt von NORMAL auf SELECTED gestellt werden. Dies besorgt die AES-Funktion objc_change.
- Wurde das Piktogramm gewählt, so kann es vorkommen, daß dadurch ein Menüeintrag wählbar ist, der vorher in schwacher Schrift erschien. Dieser Menüeintrag muß mittels der AES-Funktion menu_ienable in Normalschrift erscheinen.

l) Erzeugen eines Fensters

Das AES übernimmt ein Teil der Fensterverwaltung. Alle Elemente eines Fensters, die nicht innerhalb des Fenster liegen, werden vom AES ständig auf den neuesten Stand gebracht (z.B. wenn ein Fenster nach oben gebracht wird). Die Applikation ist für das Innere des Fensters verantwortlich. Dies ist verständlich, da das AES nicht wissen kann, was die Applikation dort darstellen möchte.

Um ein Fenster erscheinen zu lassen, muß es kreiert und geöffnet werden. Beim Kreieren wird angegeben, welche Randkomponenten das Fenster besitzen soll (Schieber, Schließbox, Infozeile usw.) und welche Maximalgröße es haben darf.

Durch den Aufruf wind_create wird das Fenster erzeugt, wobei eine Fensterkennung (Wert zwischen 1 und n) zurückgegeben wird. Diese Kennung (window handle) wird bei

jedem weiteren Aufruf aus der Fensterbibliothek benötigt. Mit Kennung 0 wird der AES-Desktop bezeichnet, eine negative Kennung zeigt an, daß keine Fenster mehr zur Verfügung stehen.

m) Öffnen eines Fensters

Das Fenster erscheint beim Aufruf von wind_open, wobei die Fensterkennung und die Anfangsposition übergeben wird. Das AES zeichnet jetzt die Randkomponenten des Fensters und sendet der Applikation die Nachricht WM_REDRAW, was bedeutet, daß sie das Fensterinnere neu zeichnen muß.

n) Schiebergrößen

Das Innere eines Fensters kann ganz oder nur teilweise angezeigt werden. Stellt man sich einen Texteditor vor, so wird meist nur ein Teil des Dokumentes angezeigt, da der gesamte Text nicht in das Fenster paßt. In diesem Fall müssen die horizontalen und vertikalen Schieber (Slider) neu gezeichnet werden, was durch den AES-Befehl wind_set bewerkstelligt wird.

o) Verändern der Fenstergröße

Zieht der Benutzer an der rechten unteren Ecke eines Fensters dieses auf eine neue Größe, so passiert folgendes: Das AES sendet der Applikation die Meldung WM_SIZED mit der neuen gewünschten Größe. Die Applikation benutzt diese Größe, um festzustellen, ob sie erlaubt ist. Ist die Größe korrekt, so wird durch den wind_set-Befehl die Größe an das AES als korrekt zurückgegeben. Ansonsten wird die Applikation die neue Größe ignorieren oder sie an die nächste erlaubte Größe anpassen.

Ist das neue Fenster größer als das alte, so sendet das AES wieder die Meldung WM_REDRAW, da die jetzt freien Teile von der Applikation neu gezeichnet werden müssen.

p) Rechtecklisten

Die Applikation muß nur die Teile des Fensterinneren zeichnen, die im Augenblick sichtbar sind. Um die sichtbaren Teile eines Fensters zu bestimmen, legt das AES sogenannte Rechtecklisten an und hält diese auf dem neuesten Stand. Eine Rechteckliste besteht aus so vielen nicht überlagerten Rechtecken, wie nötig sind, um das Fensterinnere zu beschreiben. Ist das Fensterinnere ganz zu sehen, so besteht die Rechteckliste aus einem Element, nämlich dem Fensterinneren. Ist es durch ein anderes Fenster verdeckt, so besteht die Liste mindestens aus 2 Elementen.

Um das Fensterinnere zu restaurieren, holt man sich in einer Schleife diese Rechtecke durch wind_get-Aufrufe und den Konstanten WF_FIRSTXYWH (erstes Rechteck) und

WF_NEXTXYWH für alle weiteren Rechtecke. Dann setzt man das Clipping (vs_clip) auf dieses Rechteck und zeichnet das Fensterinnere neu. Dies wird so lange durchgeführt, bis kein Rechteck mehr zurückgeliefert wird (Breite und Höhe = 0).

Vor dem Restaurieren des Fensterinhaltes, also dem Zeichnen auf den Bildschirm, muß dem AES unbedingt mitgeteilt werden, daß dieser Vorgang jetzt ansteht. Durch den Aufruf wind_update mit dem Wert BEG_UPDATE wird nur noch unserer Applikation erlaubt, den Bildschirm zu ändern. Der Bildschirmmanager sowie die Desk-Accessories sind gesperrt, damit das Zeichnen nicht unterbrochen wird.

q) Fenster aktivieren

Wurde auf ein inaktives Fenster geklickt, so sendet der Bildschirmmanager die Meldung WM_TOPPED mit der Fensterkennung. Die Applikation entscheidet nun, ob das Fenster nach oben gebracht werden darf. Ist dies der Fall, so genügt ein wind_set-Aufruf mit der Fensterkennung und dem Wert WF_TOP. Das AES bringt das Fenster nach oben und sendet die Nachricht WM_REDRAW, die von der Applikation wie üblich abgehandelt werden muß.

r) Schließen und Löschen von Fenstern

Hat der Benutzer auf die Schließbox eines Fensters geklickt, so sendet das AES die Meldung WM_CLOSED. Darf das Fenster geschlossen werden, so muß die Applikation die Funktion wind_close aufrufen. Das Fenster wird dann vom Bildschirm gelöscht. Das Fenster kann jederzeit wieder geöffnet werden, da die Fensterkennung noch nicht freigegeben wurde. Dies wird mit dem Aufruf wind_delete gemacht. Jetzt ist die Kennung wieder frei (z.B. für Fenster von Accessories).

Zusammenfassung: Das letzte Kapitel hat gezeigt, wie die Aufruffolgen aussehen können, wenn eine GEM-Applikation läuft. Diese Folgen sind mehr oder weniger bei allen Applikationen gleich. Es sieht auch so aus, als ob sehr viel Aufwand getrieben werden muß, um ein System mit Bildschirmfenstern zu entwickeln. Dies ist in der Tat so. Ziel dieses Buches ist es aber, dem Entwickler so viel wie möglich Arbeit abzunehmen, damit er sich auf das Wesentliche konzentrieren kann. So ist z.B. das Modul WINDOWS.C des SCRAP-Programmes eine vollständige Fensterverwaltung, an der nichts mehr geändert werden muß. Der Entwickler muß praktisch nur noch die Funktion zur Verfügung stellen, die das Innere des Fensters zeichnet.

Das Gleiche gilt für die Behandlung von Resourcen. So wird die Anpassung von Piktogrammen an verschiedene Grafikkarten über die VDI-Funktion v_trnfm vollständig in dem Modul RESOURCE.C übernommen. Auch andere Anpassungen wie benutzerdefinierte Objekte werden automatisch verwaltet.

Das Programm SHOWGEM, welches GEM-Metafiles ausgibt, kann als Mini-Beispielapplikation ohne Menüleiste zum Verständnis der Behandlung von Alert- und Dialogboxen dienen. Wer GEM aber komplett verstehen will, sollte sich das Programm SCRAP als Komplettbeispiel ansehen. Es beinhaltet alles, was ein professionelles GEM-System benötigt. Auf dieses Tool aufbauend, können perfekte GEM-Applikationen auf allen Rechnern entwickelt werden.

### 2.4.6 Allgemeine Vorgehensweise beim Aufbau eines GEM-Programms

Um ein GEM-Programm zu entwickeln, sollte man sich zunächst überlegen, welche Funktionen damit ausgeführt werden sollen. Anhand der Funktionalität kann man als erstes eine Menüleiste mit den entsprechenden Drop-Down-Menüs erstellen. Es spiegelt zu 80 Prozent die Möglichkeiten einer Applikation wieder.

Der nächste Schritt ist die Überlegung, wie die einzelnen Dialoge auszusehen haben, die per Menü- oder Piktogrammwahl erscheinen sollen. Im Programm SHOWGEM.C z.B. ist eine Dialogbox für alle Einstellungen und den Start der Ausgabe vorgesehen.

Das Erzeugen und Erstellen von Menüleisten und Dialogboxen geschieht im GEM interaktiv. Dazu existiert ein wichtiges Programm namens RCS oder auch Resource-Construction-Set. Verschiedene Firmen liefern mit ihren Compilern eventuell andere Versionen aus, aber es handelt sich im Endeffekt immer um das gleiche Programm. Eine ausführliche Beschreibung der Bedienung von solch einem System würde hier zu weit führen. Es gibt sie aber in dem Buch „Softwareentwicklung auf dem ATARI ST", das wir beim gleichen Verlag veröffentlicht haben.

Während der Programmentwicklung wird man auch merken, daß man diese oder jene Fehlermeldung benötigt. Auch Fehlermeldungen werden mit dem RCS erstellt.

Das Ergebnis einer RCS-Sitzung ist eine Datei, die die kompletten Resourcen einer GEM-Applikation enthält. Die Datei hat immer das Suffix RSC (Resource) und enthält Menüleisten, Dialogboxen, Fehlermeldungen, Piktogramme, Hilfemeldungen usw. Der Vorteil von solchen Resource-Dateien liegt auf der Hand; für die Übersetzung in andere Sprachen muß das Programm nicht neu geschrieben oder übersetzt werden, sondern nur die Resource-Datei mit den fremdsprachigen Texten behandelt werden.

Zu jeder Dialogbox oder Menüleiste gehört ein Objektbaum. Die Wurzel des Baumes ist die Box, alle Objekte in der Dialogbox sind Kinder dieser Box (siehe auch Object-Library). Jedem Objekt (Zeichenkette, Knopf, Piktogramm, Menütitel, Menüeintrag, editierbare Textfelder) kann im RCS ein Objektname zugewiesen werden. Diese Namen sind Konstanten z.B. für ein C-Programm. Die Konstante ist der Index des Objekts im entsprechenden Objektbaum. Um ein Objekt beim Programmieren anzusprechen, benötigt man die Wurzel des Objekts und den Index (die Objektnummer im Baum). Diese Nummern werden vom RCS automatisch vergeben, und man braucht sich nicht um sie

zu kümmern. Man gibt also nur den Objekten einen Namen, mit denen man später arbeiten möchte. Die Überschrift einer Dialogbox z.B. muß keinen Namen im RCS bekommen, da sie sich ja nie ändert.

Die Konstanten werden in eine Datei mit Suffix „.H" geschrieben, die man in sein C-Programm inkludieren kann. Man hat dann Zugriff auf die Objekte per Namen. Über den AES-Befehl rsrc_load können die Resource-Datei geladen und die Objektbäume an Variablen zugewiesen werden. Das Programm SHOWGEM zeigt die Benutzung einer Resource-Datei und der Objektbäume in der Datei. Da es sich nur um eine kleine Applikation handelt, ist der Vorgang des Benutzens noch recht übersichtlich und für Anfänger geeignet. Eine genaue Beschreibung befindet sich in Kapitel 4.

Wenn die Resourcen geladen sind, zeigt man die Menüleiste an und wartet mit dem Befehl evnt_multi auf ein Ereignis. Wann ein Ereignis auftritt, wurde weiter oben beschrieben. Je nach Ereignis wird man ein Fenster öffnen oder z.B. eine Dialgbox anzeigen und abhandeln. Das Programm SCRAP zeigt alle Möglichkeiten von Ereignissen und deren Behandlung an.

### 2.4.7 Funktionen der AES-Gruppen

Die AES-Funktionen wurden schon in sehr vielen Büchern beschrieben. Aber der erklärende Text jeder einzelnen Funktion war meist nur 1 oder 2 Zeilen lang. Oft wurden auch die falschen Erklärungen gegeben. Es sollen an dieser Stelle alle Funktionen, die eine ausführliche Erklärung benötigen, genau besprochen werden. Funktionen, die keine große Bedeutung haben, werden nur der Vollständigheit halber aufgelistet.

Zum Nachschlagen der Parameter jeder einzelnen Funktion kann ein x-beliebiges GEM-Buch benutzt werden. Wir empfehlen das Buch aus gleichem Verlag („Softwareentwicklung auf dem ATARI ST"), das alle GEM-Funktionen bis GEM 2.X beschreibt. Parameter werden in diesem Kapitel nur dort erläutert, wo es für das Verständnis wichtig erscheint. Außerdem werden alle neuen GEM/3-Funktionen mit Parametern genau erläutert, da diese in keinem einzigen Buch für den ATARI ST erwähnt werden, aber für den einen oder anderen interessant sind, der Programme für ATARI und MS-DOS erstellen möchte.

Zu jeder Bibliothek (Library) werden in einer Tabelle kurz alle Funktionen mit ihrem Zweck aufgelistet. Vor dem C-Funktionsnamen werden die Funktionsnummern angegeben. Wir verwenden wieder die englischen Namen, um das Benutzen der Originaldokumentation zu erleichtern.

1. APPLICATION-LIBRARY

Sie wird benötigt, um alle AES-Aufrufe zu initialisieren. Außerdem kann man Nachrichten zwischen Applikationen austauschen. Vor der Beschreibung der Funktion soll noch der Begriff „Nachricht" erläutert werden.

Eine Nachricht befindet sich in einem Feld mit Grundtyp WORD (also 16 Bit). Das Feld ist 8 Elemente groß. In C-Schreibweise also:

```
GLOBAL WORD msgbuff [8];
```

Dieses Feld wird von der Applikation gefüllt, falls es Nachrichten senden möchte. Beim Lesen von Nachrichten wird es vom AES gefüllt. Die einzelnen Felder haben folgende Bedeutung:

msgbuf [0] = Typ der Nachricht
msgbuf [1] = Applikationskennung (ap_id) des Absenders
msgbuf [2] = −1
msgbuf [3] = Länge der Nachricht in Bytes
msgbuf [4] − msgbuf [5] = Adresse des Nachrichtenpuffers
msgbuf [6] = frei verwendbar
msgbuf [7] = frei verwendbar

| | | |
|---|---|---|
| 10 | appl_init | Applikation initialisieren |
| 11 | appl_read | Nachricht lesen |
| 12 | appl_write | Nachricht schreiben |
| 13 | appl_find | andere Applikationen finden |
| 14 | appl_tplay | AES-Funktionen aufnehmen |
| 15 | appl_trecord | AES-Funktionen abspielen |
| 16 | appl_bvset | Disk-Info setzen |
| 17 | appl_yield | Prozeß umschalten |
| 19 | appl_exit | Applikation beenden |

- APPL_INIT
- APPL_EXIT

Der appl_init-Aufruf muß vor allen anderen AES-Aufrufen gemacht werden. Er initialisiert die internen Strukturen des AES. Außerdem wird das Feld global (s.o.) mit Informationen gefüllt.

Beim Aufruf wird ein interner Zähler auf 0 gestellt. Dieser Zähler wird bei jedem weiteren AES-Aufruf inkrementiert. Erreicht der Zählerstand 10, so erfolgt das erste Umschalten auf einen anderen Prozeß. Dies hat zur Konsequenz, daß die Applikation innerhalb der ersten 10 AES-Aufrufe die dringlichsten Sachen erledigen muß. Dazu gehört vor allen Dingen das dynamische Reservieren von Speicher und das Laden von Resource-Dateien. Wird dies nicht gemacht, so können andere Prozesse dazwischenkommen. Fordern diese vom Betriebssystem ebenfalls Speicher an, so sind alle Speicherblöcke zerrissen. Diesen Umstand kann man besonders bei Accessories beobachten.

Durch den Aufruf appl_exit wird dem AES mitgeteilt, daß unsere Applikation jetzt beendet ist. Weitere AES-Aufrufe sind dann nicht mehr möglich.

– APPL_READ

Hiermit wird eine Nachricht gelesen. Man übergibt die Kennung der Applikation, für die die Nachricht gelesen werden soll, also unserer eigenen Applikation.

– APPL_WRITE

Dies ist das Gegenstück zu appl_read. Man kann eine Nachricht an eine andere Applikation senden (z.B. an ein Desk-Accessory wie den Druckerspooler). Dazu muß man die Kennung (ap_id) der Empfängerseite herausfinden. Man erhält sie mit einem appl_find-Aufruf. Es ist also möglich, mit anderen Programmen im System elegant zu kommunizieren. Eigene Message-Typen sollten mit Nummern von 1000 bis 32000 numeriert sein.

– APPL_FIND

Um die Kennung (ap_id) von anderen Applikationen, die gerade laufen, zu erfahren, muß die Funktion appl_find aufgerufen werden. Man übergibt den Dateinamen (ohne Suffix) und füllt bis zu 8 Zeichen auf, falls dieser kürzer ist.

Beispiel:

```
msgbuf [0] = gl_apid;
...

scrap_id = appl_find (''SCRAP    '');  /* 3 Leerzeichen nach SCRAP*/
if (scrap_id > 0) appl_write (scrap_id, 16, msgbuf);
```

Das Beispiel sucht die Applikation SCRAP. Dies könnte ein Accessory mit Namen SCRAP.ACC sein (siehe auch Kap. 6). Falls die Applikation gefunden wurde, wird eine Nachricht von 16 Bytes gesendet. Der Puffer msgbuf muß vorher gefüllt worden sein. Das nullte Element enthält unsere eigene Applikationskennung.

– APPL_TPLAY
– APPL_TRECORD

Mit diesen beiden Funktionen können AES-Ereignisse aufgezeichnet und wiedergegeben werden. Man übergibt einen Puffer und die Anzahl der Ereignisse, die aufgezeichnet werden sollen.

Ein Ereignis besteht aus 6 Bytes, einem WORD und einem LONG-Wert. Dies gilt aber nur im INTEL-Modell. Im Motorola MC68000-Modell (z.B. ATARI ST) besteht ein Ereignis aus 2 LONG-Werten! Dies wird in allen uns bekannten Büchern für GEM auf dem

ATARI ST falsch beschrieben. Da diese Funktion relativ unwichtig ist, werden wir sie auch nicht näher beschreiben.

– APPL_BVSET (ab GEM 2.x)

Hiermit können Informationen über die Disketten- und Festplattenlaufwerke gesetzt werden. Diese Informationen werden im global-Feld abgelegt (s.o). Der GEM-Desktop z.B. benutzt diesen Aufruf, wenn Diskettenlaufwerke angemeldet werden.

Soll GEM z.B. Disk-Laufwerk A: und die Plattenlaufwerke C: und D: erkennen, so muß der Aufruf wie folgt aussehen:

```
WORD bvdisk, bvhard;

bvdisk = 0x8000; /* 10000000 00000000 Bit 15 gesetzt */
bvhard = 0x3000  /* 00110000 00000000 Bit 13 und 12 gesetzt */
appl_bvset (bvdisk, bvhard);
```

– APPL_YIELD

Hiermit wird der Dispatcher veranlaßt, den Prozeß zu wechseln. Sind z.B. 2 Desk-Accessories und unsere Haupt-Applikation aktiv, und rechnet unsere Applikation in einer Schleife, ohne einen einzigen AES-Befehl zu tätigen, so kommen die beiden Accessories nicht mehr zum Zuge. Der Druckerspooler z.B. könnte keine Zeichen mehr an den Drucker senden.

Deshalb sollte bei intensivem Rechnen zwischendurch ein app_yield-Aufruf getätigt werden. Der Dispatcher läßt dann in unserem Beispiel die Accessories an die Reihe, um unsere Applikation danach fortzuführen. Der Aufruf existiert bei GEM-Version 1.X noch nicht, kann aber mit Hilfe der Datei PORTAB.H trotzdem von allen benutzt werden. Dort wird der Aufruf durch einen evnt_timer simuliert.

2. EVENT-LIBRARY

| | | |
|---|---|---|
| 20 – | evnt_keybd | Tastaturereignis abwarten |
| 21 – | evnt_button | Mausknopfereignis abwarten |
| 22 – | evnt_mouse | Mausereignis abwarten |
| 23 – | evnt_mesag | Nachrichtenereignis abwarten |
| 24 – | evnt_timer | Zeitereignis abwarten |
| 25 – | evnt_multi | Mehrfachereignis abwarten |
| 26 – | evnt_dclick | Doppelklick lesen/schreiben |

Wer schon einmal dialogorientierte Anwendungssoftware geschrieben hat, der weiß, daß man in seiner Applikation auf Eingaben des Anwenders reagieren muß. Eingaben in einem professionellen Anwendungsprogramm können von verschiedensten Quellen kommen. Der Benutzer kann die Tastatur betätigen, mit der Maus Menüs anwählen und den Mausknopf drücken. Außerdem kann er mit der Maus bestimmte Bereiche (meist Bildschirmfenster) anfahren. Die Applikation kann aber auch ohne Eingaben irgend etwas tun. So stelle man sich einen Druckerspooler vor, der einmal angestoßen wird und dann jede 1/100 Sekunde ein Zeichen an den Drucker sendet. In solch einem Fall sprechen wir von einem Zeitereignis.

Tätigt der Benutzer irgendeine der obigen Eingaben, so muß die Applikation darauf reagieren. Das Resultat einer Benutzeraktion wird auch „Ereignis" genannt. Drückt der Benutzer eine Taste, so spricht man entsprechend von einem „Tastaturereignis". In älteren Systemen wird das Warten auf verschiedene Ereignisse durch eine Schleife (Polling) realisiert, in der ständig die Tastatur, die Maus, Mausknöpfe und verstrichene Zeiten abgefragt werden. Das System war also ständig damit beschäftigt, Abfragen vorzunehmen. Dies kostet natürlich eine Menge Zeit.

Um diese Ineffizienz zu vermeiden, bietet das GEM-System die Event-Library an, die auf elegante Art das Problem löst. Die Applikation gibt an, auf welches oder auf welche Ereignisse sie warten möchte. Das Betriebssystem überwacht Tastatur, Maus etc. und gibt in dem Moment die Kontrolle an die Applikation zurück, wenn das gewünschte Ereignis aufgetreten ist. Dies hat den Vorteil, daß das System andere Prozesse laufen lassen kann (z.B. den Druckerspooler), wenn ein Ereignis, auf das man wartet, noch nicht eingetreten ist.

Wenn eine Applikation nur auf ein Ereignis (z.B. Tastatur) wartet, so kann es auf andere Ereignisse nicht reagieren und verpaßt diese sozusagen. Zu diesem Zweck kann man auch auf eine Kombination von Ereignissen warten (multiple events). Ruft man die Funktion auf, so kehrt sie zurück, wenn mindestens eines der angegebenen Ereignisse eingetreten ist. Die meisten GEM-Programm benutzen also die Funktion evnt_multi, um auf mehrfache Ereignisse zu warten.

Folgende Ereignisse werden von der Event-Library unterstützt (Parameter und Werte der Argumente lesen Sie bitte in irgendeinem GEM-Funktionshandbuch nach):

- EVNT_KEYBD

Durch den Aufruf von evnt_keybd bekommt man Informationen über eine Taste, die gedrückt wurde, zurück. Sowohl der ASCII-Code (niederwertiges Wort) als auch der Scan-Code (höherwertiges Byte) werden zurückgeliefert. Um den Status der Shift- und Alternate-Tasten zu bestimmen, muß die Funktion graf_mkstate benutzt werden.

Beispiel:

```
key = evnt_keybd ();
ascii_code = key & 0x00FF; /* untere 8 Bits */
scan_code  = key >> 8;     /* obere 8 Bits  */
```

– EVNT_BUTTON

Hiermit kann die Applikation auf das Drücken oder Loslassen von Mausknöpfen warten. Auch kann auf das Ereignis Doppelklick gewartet werden. Zusätzlich kann man angeben, ob auf das Drücken oder auf das Loslassen von Mausknöpfen gewartet werden soll. Theoretisch werden bis zu 16 Mausknöpfe unterstützt. Ein Ereignis tritt aber nur ein, wenn mindestens der linke Mausknopf gedrückt wird.

– EVNT_MOUSE

Werden in einer Applikation verschiedene Bildschirmfenster benutzt, so kommt es vor, daß für die verschiedenen Fenstertypen das Aussehen der Maus geändert werden soll. In einer Textverarbeitung könnte die Maus von einem Pfeil zu einem Textcursor (vertikale Linie) verändert werden. In einem Grafikprogramm benutzt man häufig ein Fadenkreuz. In der Beispielapplikation SCRAP wird z.B. die Mausform des Textfensters geändert.

Beim Aufruf der Funktion wird das Rechteck übergeben, bei dessen Ein- oder Austritt der Maus ein Ereignis stattfinden soll.

– EVNT_MESAG

Wie schon weiter oben erwähnt, besteht das GEM-System aus mehreren Prozessen (Benutzerprogramm, Accessories, Bildschirmmanager). Diese Prozesse müssen untereinander Nachrichten austauschen können. Einer der wichtigsten Prozesse ist der Bildschirmmanager. Er wird immer dann aktiv, wenn die Maus in die Menüzeile bewegt wurde oder mit den Randkomponenten eines Fensters Aktionen durchgeführt wurden (z.B. Verschieben/Vergrößern eines Fensters).

Der Bildschirmmanager sendet der Applikation, zu der das oberste, aktive Fenster gehört, Nachrichten. Diese sogenannten Standardnachrichten können mit der Funktion evnt_mesag gelesen werden. Die Nachrichten haben immer eine Länge von 16 Bytes und sind in Worten organisiert. In C-Schreibweise könnte man sie wie folgt definieren:

WORD msgbuf [8];

Die einzelnen Felder haben folgende Bedeutung:

msgbuf [0] = Typ der Nachricht
msgbuf [1] = Applikationskennung (ap_id) des Absenders
msgbuf [2] = Länge der Nachricht (ohne die 16 Bytes). Ist die Nachricht länger als 0 (> 0) so muß mit der Funktion appl_read der Rest der Nachricht gelesen werden.
msgbuf [3] – msgbuf [7] = Abhängig vom Typ der Nachricht

Entsprechend dem Typ der Nachricht muß die Applikation reagieren. Wurde z.B. ein Menüpunkt angeklickt, so sollte zum Menü-Handler gesprungen werden, der den Rest der Nachricht verarbeitet (siehe Beispielprogramm SCRAP).

Folgende Standardmeldungen können in msgbuf [0] auftreten:

a) MN_SELECTED

Der Benutzer hat einen Menüpunkt mit der Maus angewählt. Die Feldelemente msgbuf [3] bzw. msgbuf [4] enthalten den Index des Menüpunktes im Objektbaum der Menüleiste.

b) WM_REDRAW

Durch Fensteraktionen ist ein Zustand eingetreten, der dazu führt, daß der Fensterinhalt der Applikation neu gezeichnet werden muß. Man bekommt die Kennung des Fensters sowie Position, Breite und Höhe des Fensters zurück.

c) WM_TOPPED

Ein Fenster wurde aktiviert (nach oben gebracht). Zurückgegeben wird die Kennung (window handle) des Fensters.

d) WM_CLOSED

Ein Fenster wurde geschlossen. Es wird durch die Kennung identifiziert.

e) WM_FULLED

Wenn der Benutzer auf die Vergrößerungsbox eines Fensters geklickt hat, wird diese Meldung zusammen mit der Kennung des Fensters vom Bildschirmmanager an die Applikation gesendet.

f) WM_ARROWED

Klickt der Benutzer auf einen der Pfeile in einem Fenster oder den Bereich der Schieber, so wird diese Nachricht gesendet. Im msgbuf [4] steht dann genau, wohin der Benutzer geklickt hat.

g) WM_HSLIDE

Diese Nachricht wird gesendet, wenn der Benutzer den horizontalen Schieber eines Fensters bewegt hat.

h) WM_VSLIDE

Diese Nachricht wird gesendet, wenn der Benutzer den vertikalen Schieber eines Fensters bewegt hat.

i) WM_SIZED

Wurde vom Benutzer die Größe eines Fensters verändert, so wird diese Nachricht an die Applikation gesendet. Die neue Breite und Höhe des Fensters werden ebenfalls in der Nachricht angegeben.

j) WM_MOVED

Wurde vom Benutzer die Position eines Fensters verändert, so wird diese Nachricht an die Applikation gesendet. Die neuen x- und y-Werte des Fensters werden ebenfalls in der Nachricht angegeben.

11. WM_UNTOPPED

Wird das oberste Fenster durch das Nachobenbringen eines anderen Fensters inaktiv, so wird diese Nachricht gesendet. Die Applikation hat dann noch die Möglichkeit das Innere des Fensters zu retten, falls dies nötig ist (z.B. bei einem Grafikprogramm, das die erzeugte Grafik zunächst im Bildschirm puffert).

12. AC_OPEN

Klickt ein Benutzer in das Accessory-Menü und wählt damit ein Accessory aus, so wird diese Meldung an alle Accessories gesendet. Durch den Bezeichner, der mitgesendet wird, kann entschieden werden, ob die Meldung für Ihr Accessory bestimmt ist.

13. AC_CLOSE

Diese Meldung wird gesendet, wenn die Hauptapplikation verlassen wird oder der Bildschirm gelöscht werden soll. Das Accessory sollte dann sämtliche Fenster freigeben, die es geöffnet hatte.

14. Andere Meldungen

Im GEM-System unter MS-DOS existiert das Accesory CALCLOCK.ACC. Es beinhaltet einen Taschenrechner, eine Uhr und einen Druckerspooler. Der Druckerspooler druckt Dateien im Hintergrund. Die Dateien müssen fertig formatiert sein und werden Byte für Byte vom Spooler an den Drucker gesendet. Das Programm OUTPUT benutzt den Spooler, wenn man „Druck auf Datei“ anwählt. Man kann dort „Im Hintergrund drucken“ angeben. Das OUTPUT sendet dann eine Nachricht an den Spooler, die wie folgt aussieht:

msgbuf [0] = 100
msgbuf [1] = Applikationskennung (ap_id) des Absenders
msgbuf [2] = −1
msgbuf [3] = Länge des Dateinamens
msgbuf [4] − msgbuf [5] = Adresse des Dateinamens
msgbuf [6] = Anzahl der Kopien
msgbuf [7] = 0: Datei nach Drucken nicht löschen
1: Datei nach Drucken löschen

Der Spooler sendet nach dem Empfangen der Nachricht die Quittung an den Sender, die wie folgt aussieht:

msgbuf [0] = 101;
msgbuf [1] = Kennung des Spooler (ap_id)
msgbuf [2] − msgbuf [7] = 0;

Beispiel: Wir wollen die Datei DRUCK.TXT auf der Scrapdirectory über den Systemspooler ausdrucken. Der Systemspooler ist ein Accessory mit Dateinamen CALCLOCK.ACC.

```
BYTE filename [80];
WORD ap_id, spool_id;
WORD msgbuf [8];


ap_id    = appl_init (); /* eigene Kennung */
spool_id = appl_find ("CALCLOCK");
scrp_read (filename);
strcat (filename, "DRUCK.TXT");


if (spool_id >= 0)
{
  msgbuf [0] = 100;
  msgbuf [1] = ap_id;
  msgbuf [2] = -1;
  msgbuf [3] = strlen (filename);
  msgbuf [4] = (WORD)((LONG)filename & 0xFFFF);  /* low word */
  msgbuf [5] = (WORD)((LONG)(filename) >> 16);   /* high word */
  msgbuf [6] = 1;
  msgbuf [7] = 1;


  appl_write (spool_id, 16, msgbuf);
  evnt_mesag (msgbuf);
  ok = msgbuf [0] == 101;
} /* if */
```

In obigem Beispiel wird zuerst unsere eigene Applikation initialisiert (appl_init), wobei wir unsere Kennung (ap_id) erhalten. Danach suchen wir die Kennung des Spoolers CALCLOCK. Ist diese Kennung größer oder gleich 0, so können wir dem Spooler eine Nachricht senden. Der msgbuf wird entsprechend vorbelegt. Durch appl_write wird die Nachricht gesendet. Wir warten dann sofort auf die Quittung des Spoolers (evnt_mesag). Die Quittung ist korrekt, wenn msgbuf [0] den Wert 101 hatte.

Achtung! Das oben beschriebene Handshaking (appl_write, evnt_mesag) funktioniert nur, wenn der Bildschirmmanager seine Arbeit verrichten kann, da er ja als parallel laufender Prozeß alle Aktivitäten überwacht. Er ist dann aktiv, wenn die Funktion wind_update (END_UPDATE) aufgerufen wurde, falls vorher ein wind_update (BEG_UPDATE) gemacht wurde. Außerdem müssen bei einem 68000-System die Werte msgbuf [4] und [5] vertauscht werden.

Falls ein Softwarehaus oder eine Privatperson einen Systemspooler für den ATARI ST entwickeln möchte, so halte es sich bitte strikt an o.a. Nachrichtenprotokoll und an den Programmnamen (CALCLOCK.ACC). Dann sind alle Programme, die diesen Spooler benutzen, kompatibel zu den verschiedenen GEM-Versionen.

– EVNT_TIMER

Manchmal ist es nötig, eine bestimmte Zeit warten zu müssen (z.B. 2 Sekunden). Dies kann man natürlich durch ständiges Auslesen der Rechneruhr bewerkstelligen. Dadurch wird aber wertvolle Rechenzeit vergeudet. Ein Aufruf von evnt_timer verhindert dies. Man gibt an, wieviele Millisekunden man warten möchte und bekommt nach Ablauf der Zeit die Nachricht vom Typ MU_TIMER in den Nachrichtenpuffer.

– EVNT_MULTI

Wie schon weiter oben erwähnt, ist es oft nötig, auf mehrere Ereignisse gleichzeitig warten zu müssen, um keines der Ereignisse zu verpassen. Zu diesem Zweck gibt es die AES-Funktion evnt_multi. Dort kann man auf eine Kombination aller eben beschriebenen Ereignisse warten. Deshalb ist die Funktion evnt_multi die zentrale Warteschleife auf Benutzeraktionen in einem Dialogsystem. Sie findet sich in jedem größeren GEM-Programm und sollte immer benutzt werden.

– EVNT_DCLICK

Durch Aufruf von evnt_dclick kann die Geschwindigkeit eingestellt werden, innerhalb der ein Doppelklick der Maus durchgeführt werden muß. Die Werte reichen von 0 (langsam) bis 4 (schnell).

3. MENU-LIBRARY

| | |
|---|---|
| 30 – menu_bar | Menüzeile anzeigen/löschen |
| 31 – menu_icheck | Check-Häkchen setzen/löschen |
| 32 – menu_ienable | Menüeinträge grau/normal darstellen |
| 33 – menu_tnormal | Menüeinträge invers/normal darstellen |
| 34 – menu_text | Text in Menüeintrag ändern |
| 35 – menu_register | Desk-Accessory anmelden |
| 36 – menu_unregister | Desk-Accessory abmelden |
| 37 – menu_click | Einstellung von Drop-Down/Pull-Down-Menüs |

Menüzeilen erscheinen am oberen Bildschirmrand eines GEM-Programmes. Sie können auch in einem Fenster erscheinen (durch Benutzen des Programmes SCRAP), was weitere Aktionen ermöglicht. Menüzeilen werden mit dem RCS erstellt und können nach dem Laden (rsrc_load) mit einem einzigen Befehl installiert werden (menu_bar).

Ab diesem Zeitpunkt übernimmt der Bildschirmmanager die Verwaltung der Menüzeile. Fährt ein Benutzer mit der Maus in die obere Bildschirmzeile, so läßt der Manager das Menü herunterklappen, wobei der Benutzer einzelne Menüpunkte anfahren und anklicken kann. Tut er dies, so sendet der Bildschirmmanager die Nachricht, welches Menü angewählt wurde. Dazu befinden sich im Nachrichtenpuffer der Typ MN_SELECTED sowie die Indizes des Drop-Down-Menüs und des angeklickten Menüeintrages. Diesen Indizes wurden im Resource-Construction-Set Namen gegeben (Konstanten in C), so daß man vom Programm aus mit den C-Namen arbeiten kann.

Menüeinträge, die nicht angewählt werden sollen, können grau (disabled) dargestellt werden. Links neben Menüeinträgen können Häkchen (check marks) gesetzt und gelöscht werden.

– MENU_BAR

Damit wird die Menüzeile für eine Applikation (oder für ein Desk-Accessory ab GEM 2.X) eingerichtet. Die Menüzeile wird sichtbar und kann vom Benutzer angefahren werden. Man kann die Menüzeile auch wieder entfernen, was vor dem Beenden der Applikation geschehen sollte.

– MENU_ICHECK

Häkchen vor Menüeinträgen können mit dieser Funktion gesetzt und wieder gelöscht werden. Man muß selbst darauf achten, daß man beim Erstellen der Menüeinträge (am besten) 2 Leerzeichen vor jeden Menüeintrag setzt, um Platz für ein Häkchen zu reservieren.

– MENU_IENABLE

Sollen Menüeinträge nicht anwählbar sein, so muß dies vom Bildschirmmanager erkannt werden. Dazu benutzt man die Funktion menu_ienable, die Menüeinträge grau (in schwacher Schrift) erscheinen läßt. Soll ein Menüpunkt wieder anwählbar sein, so kann dies ebenfalls mit der gleichen Funktion bewerkstelligt werden.

– MENU_TNORMAL

Wurde ein Menüpunkt angewählt, so erscheint die dazugehörige Menüüberschrift in inverser Schrift. Dies wird vom Bildschirmmanager erledigt. Nach dem Ausführen der Funktion, die vom Benutzer angewählt wurde, muß die Überschrift wieder auf Normalschrift geändert werden. Gibt der Benutzer einen Befehl über Tastatur ein, und entspricht dieser Befehl dem Anwählen eines Menüpunktes, so sollte die entsprechende Überschrift invers dargestellt werden. Ein Aufruf von menu_tnormal übernimmt das inverse und normale Anzeigen von Menüüberschriften.

– MENU_TEXT

Mit dieser Funktion kann der Text von Menüeinträgen zur Laufzeit geändert werden. Man gibt den Objektbaum, die Objektnummer sowie die Adresse des neuen Textes an. Der neue Text darf nicht länger als der alte sein, da sonst Speicher überschrieben wird.

Interessant ist folgende Tatsache: Man kann auch Texte von Desk-Accessories ändern, indem man in der Variablen tree nicht die Adresse des Objektbaumes übergibt, sondern im höherwertigen Wort eine 0 und im niederwertigen Wort die Prozeßkennung (process id) übergibt, die man durch einen Aufruf von menu_register erhält.

– MENU_REGISTER
– MENU_UNREGISTER

Hiermit kann ein Desk-Accessory angemeldet werden. Man übergibt die Kennung der Applikation (durch appl_init) und die Zeichenkette, die im Menüeintrag erscheinen soll. Man erhält eine Kennung von 0 bis 5 (6 Accessories sind möglich), mit der man sein eigenes Desk-Accessory identifizieren kann.

Um ein Desk-Accessory aus der Menüleiste zu entfernen, wird die Funktion menu_unregister benutzt, die ab GEM 2.X zur Verfügung steht. Viel Sinn hat dies nicht, da der Speicherplatz, den das Accessory belegt, ja nicht mehr freigegeben wird.

– MENU_CLICK

Ab GEM/3 steht diese Funktion zur Verfügung. Damit kann eingestellt werden, ob man Drop-Down-Menüs (klappen herunter, wenn die Maus in die Menüzeile kommt) oder Pull-Down-Menüs (klappen erst herunter, wenn man mit der Maus in einen Menütitel klickt) haben möchte.

Zwei Parameter müssen übergeben werden:

click: FALSE = Drop-Down-Menüs
TRUE = Pull-Down-Menüs

setit: FALSE = Einstellung abfragen
TRUE = Neue Einstellung setzen

Beispiel: Die Applikation bestimmt Pull-Down-Menüs.

```
click = TRUE;
setit = TRUE;
menu_click (click, setit);
```

4. OBJECT-LIBRARY

| | |
|---|---|
| 40 – objc_add | Objekte an Baum anhängen |
| 41 – objc_delete | Objekte aus Baum löschen |
| 42 – objc_draw | Objek oder Baum zeichnen |
| 43 – objc_find | Bestimmung Maus/Objektposition |
| 44 – objc_offset | Objektposition errechnen |
| 45 – objc_order | Objekte im Baum bewegen |
| 46 – objc_edit | Text in Objekt editieren |
| 47 – objc_change | Status eines Objektes ändern |

Der Aufbau eines Objektbaumes wurde vor allem in unserem Buch „Softwareentwicklung auf dem ATARI ST“ ausführlichst erläutert. Aus diesem Grunde werden wir nur noch einmal kurz auf Objektbäume eingehen und nur die Dinge erläutern, die neu sind oder in keinem Buch gut erklärt wurden.

Objektbäume werden mit dem Resource-Construction-Set erstellt. Sie können z.B. Menüzeilen und Dialogboxen sein, also all diejenigen Elemente außer Bildschirmfenstern, die Sie in allen GEM-Programmen schon einmal kennengelernt haben. In einem Baum gibt es meist eine Wurzel und Objekte unter dieser Wurzel (sogenannte Kinderobjekte). Jedes der Kinderobjekte kann wieder Kinderobjekte haben. So entsteht eine Hierarchie wie in Abbildung 2.16.

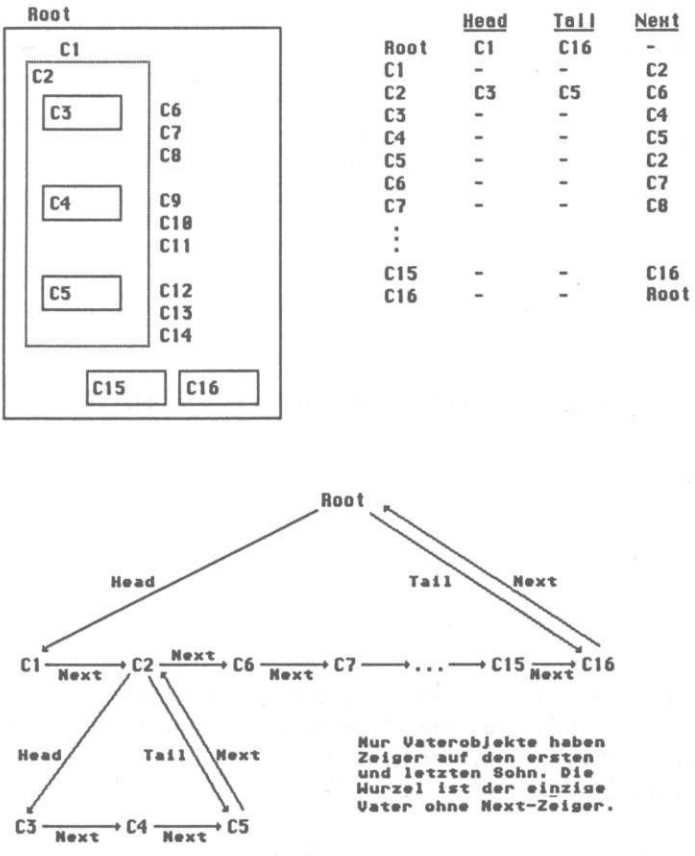



**Abb. 2.16:** Struktur eines Objektbaumes

In der Abbildung sieht man z.B. eine Dialogbox mit 17 Objekten. Die weiße Hintergrundbox ist die Wurzel des Baumes (ROOT). Darin befinden sich 16 Objekte (C1 – C16). C2 selbst ist wieder eine Box, die 3 Objekte enthält (C3 – C5). Die Objekte C6 bis C14 könnten z.B. Zeichenketten oder Piktogramme sein. Die beiden Objekte C15 und C16 könnte man sich als OK- und Abbruch-Knöpfe vorstellen.

Objekte werden intern in einem Feld vom Typ OBJECT (siehe AES.H) abgelegt. In C-Schreibweise könnte man einen Objektbaum wie folgt definieren:

```
typedef struct object
{
  WORD  ob_next;   /* -> object's next sibling      */
  WORD  ob_head;   /* -> head of object's children  */
  WORD  ob_tail;   /* -> tail of object's children  */
```

```
  UWORD ob_type;   /* type of object- BOX, CHAR, ... */
  UWORD ob_flags;  /* flags                          */
  UWORD ob_state;  /* state- SELECTED, CROSSED, ...  */
  LONG  ob_spec;   /* "out"- -> anything else        */
  WORD  ob_x;      /* upper left corner of object    */
  WORD  ob_y;      /* upper left corner of object    */
  WORD  ob_width;  /* width of object                */
  WORD  ob_height; /* height of object               */
} OBJECT;

OBJECT *dialog;
```

Die Elemente des Feldes sind die einzelnen Objekte. So steht das Wurzelobjekt an erster Stelle:

Wurzel = dialog [0]
1. Kind = dialog [1]
n. Kind = dialog [n]

usw.

Jedem Objekt kann im Resource-Construction-Set ein Name gegeben werden, wenn dies nötig ist. So wird man dem Wurzelobjekt in obigem Beispiel den C-Namen (Konstante) DIALOG geben, während man etwa dem OK-Knopf den Namen DOK (Dialog OK), dem Abbruch-Knopf den Namen DCANCEL gibt.

Die Variable dialog ist zu diesem Zeitpunkt noch nicht gefüllt. Man tut dies, nachdem man seine Objekte geladen hat (rsrc_load). Anschließend kann man mit dem Namen der Objekte auf jedes einzelne Objekt zugreifen.

Beispiel:

```
appl_init ();
rsrc_load ("SHOWGEM.RSC");
rsrc_gaddr (RTREE, DIALOG, &dialog);
```

Die Variable dialog, die auf das Wurzelobjekt unseres Objektbaumes zeigt, ist jetzt gesetzt. Auf jedes einzelne Objekt dieses Baumes kann z.B. wie folgt zugegriffen werden:

```
status = dialog [DCANCEL].ob_state;
```

Jedes Objekt wird durch einige Werte gekennzeichnet, so daß sich ein Objektbaum als Feld von Strukturen (array of record) ergibt. Wie man an obiger Definition des Typs OBJECT sieht, sind die Objekte über die Felder ob_next, ob_head und ob_tail miteinander verzeigert, was aber meist nicht interessiert.

Im Feld ob_type steht der Typ des Objekts (siehe AES.H). Dort wird z.B. G_BUTTON (Knopf), G_STRING (Zeichenkette), G_ICON (Piktogramm) usw. eingetragen.

Das Feld ob_flags sagt, ob das Objekt selektiert werden darf (SELECTABLE), ob es versteckt ist (HIDETREE) und vieles mehr.

Im Feld ob_state wird der Status eines Objektes abgelegt. Dort befinden sich Werte wie „es wurde selektiert" (SELECTED), „es ist grau dargestellt" (DISABLED) usw.

In den Feldern ob_x, ob_y, ob_width und ob_height werden die Koordinaten des Objektes relativ zum Vaterobjekt bzw. zum Bildschirm (wenn es sich um das Wurzel- oder Root-Objekt handelt) abgelegt. Nach dem Laden durch rsrc_load sind diese Werte Rasterkoordinatenwerte. Solange sie sich in der Resource-Datei befinden, werden die Werte durch Vielfaches von Einzelzeichen (character width) gekennzeichnet.

Das Feld ob_spec hat eine ganz besondere Bedeutung. Je nach Typ des Objektes kann der Inhalt variieren. Ist der Typ z.B. G_STRING, also eine Zeichenkette, so zeigt der ob_spec-Wert auf die Zeichenkette, da sie selbst ja nicht in der Objekt-Struktur untergebracht werden kann. Ist der Typ z.B. G_ICON, also ein Piktogramm, so zeigt der ob_spec-Wert auf die Beschreibung des Piktogrammes (ICONBLK-Struktur) usw. Die Strukturen, auf die ob_spec verweisen, sind in der Datei AES.H definiert. In der folgenden Tabelle werden zu jedem Objekttyp die ob_spec-Werte erläutert:

| ob_type | ob_spec-Wert |
|---|---|
| G_BOX | Farbe und Breite des Randes |
| G_TEXT | Zeiger auf TEDINFO-Struktur |
| G_BOXTEXT | Zeiger auf TEDINFO-Struktur |
| G_IMAGE | Zeiger auf BITBLK-Struktur |
| G_USERDEF | Zeiger auf APPLBLK_Struktur |
| G_IBOX | Farbe und Breite des Randes |
| G_BUTTON | Zeiger auf Zeichenkette |
| G_BOXCHAR | Zeichen, Farbe und Breite des Randes |
| G_STRING | Zeiger auf Zeichenkette |
| G_FTEXT | Zeiger auf TEDINFO-Struktur |
| G_FBOXTEXT | Zeiger auf TEDINFO-Struktur |
| G_ICON | Zeiger auf ICONBLK-Struktur |
| G_TITLE | Zeiger auf Zeichenkette |

Der Objektstatus ob_state gibt an, ob z.B. ein Objekt selektiert wurde (SELECTED). Durch die einfache Abfrage

```
if (dialog [KNOPF].ob_state & SELECTED)
{
  ...weitere Aktionen
}
```

kann z.B. geprüft werden, ob ein Objekt mit der Maus angewählt wurde. In obigem Beispiel wurde ein Objekt mit Namen KNOPF selektiert.

Neue Objektstati ab GEM 2.X sind:

WHITEBAK ($0 \times 0040$) und
DRAW3D ($0 \times 0080$)

Beide Stati werden nur bei Objekten vom Typ G_ICON benutzt. Ist WHITEBAK gesetzt und befindet sich das Piktogramm auf einer weißen Fläche, so wird die Maske des Piktogrammes nicht gezeichnet. Durch Setzen des Status spart sich das VDI bei der Ausgabe wertvolle Zeit.

Sind die beiden Bits DRAW3D und SELECTED im Objektstatus gesetzt, so wird die Maske 3-mal gezeichnet. Einmal an der angegebenen Position und jeweils 1-mal um 1 Pixel nach links oben und rechts unten versetzt. Es ergibt sich dann eine leicht dreidimensionale Darstellung.

– Benutzerdefinierte Objekte

An dieser Stelle soll eine wichtige Erweiterungsmöglichkeit für Objekttypen erklärt werden. Es handelt sich um benutzerdefinierte Objekte. Sie sind vom Typ G_USERDEF und wurden von den GEM-Systementwicklern eingebaut, um dem AES ein Hintertürchen für Erweiterungen frei zu halten. Jeder kennt die Standardobjekte aus Dialogboxen: Zeichenketten, Knöpfe, Piktogramme, Radio-Knöpfe usw. Solche Objekte können im Resource-Construction-Set erzeugt und abgespeichert werden. Der Vorteil einer Dialogbox liegt ganz einfach darin, sie mit einem Befehl zeichnen zu können (objc_draw) und das gesamte Handling durch den AES-Befehl form_do abhandeln zu lassen. Man braucht sich dann bei der Eingabe in eine Dialogbox um nichts mehr zu kümmern.

Nun reichen uns die Standardobjekte, die vom AES angeboten werden, aber nicht. Wir möchten Dialogboxen mit runden Knöpfen und ankreuzbaren Kästchen haben. Solche Objekte werden auf einem Apple Macintosh und unter Microsoft-Windows bzw. dem Presentation Manager ja auch angeboten. Zu diesem Zweck legen wir uns einfach neue Typen an. Im Feld ob_type eines OBJECT-Typs wird vom AES nur das untere Byte benutzt. Das höherwertige Byte kann von der Applikation frei belegt werden. Wir haben also die Möglichkeit, 255 neue Objekttypen zu definieren.

Wir benutzen im SCRAP-Programm für 3 neue Objekttypen aber die unteren 5 Bits nicht, da sie bei einer neuen Eingaberoutine (form_do) gebraucht werden. So bleiben 3 Bits

übrig, mit denen 8 neue Objekttypen definiert werden können. 3 davon sind:

```
#define DHEADER      0x20
#define DCHECKBOX    0x40
#define DRBUTTON     0x60
```

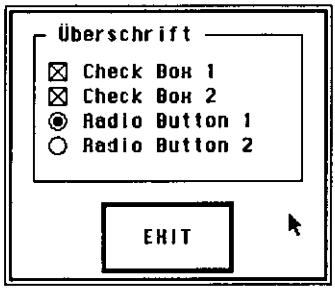



**Abb. 2.17:** Dialogbox mit neuen Objekttypen

DHEADER ist eine Überschrift, die z.B. mit einem Rahmen am besten benutzt werden kann. Objekte von diesem Typ werden im Objektbaum um eine halbe Zeichensatzhöhe nach oben geschoben.

DCHECKBOX sind ankreuzbare Boxen. Ist das Objekt selektiert (SELECTED), so erscheint eine gekreuzte Linie in einem Rahmen. Ist es nicht selektiert, erscheint nur der Rahmen. Sowohl der Rahmen der Linie (Linienfarbe) als auch das innere Kreuz (Füllmusterfarbe) können im RCS farblich getrennt eingestellt werden (z.B. grüne Box mit rotem Kreuz).

DRBUTTON sind runde Radio-Knöpfe wie auf einem Apple-Macintosh. Es sind kleine Piktogramme, die an der entsprechenden Stelle gezeichnet werden. Rund sind die Knöpfe allerdings nur, wenn der Bildschirm in etwa ein Auflösungsformat von 1:1 hat, wie dies beim ATARI ST der Fall ist. In einer Auflösung von 640x200 z.B. sind die Knöpfe auch rund, da dort entsprechend verzerrte Piktogramme (RBLNORM, RBLSEL) benutzt werden. Auch sie können farbig sein. Dazu stellt man das Füllmuster im RCS auf die entsprechende Farbe ein.

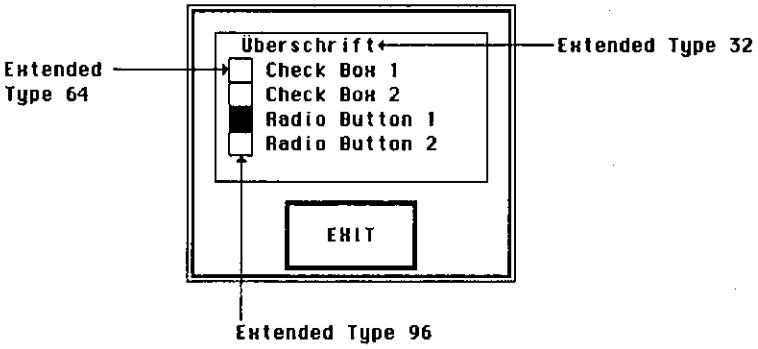



**Abb. 2.18:** Herstellung der neuen Objekttypen

Die Frage erhebt sich, wie man solche Objekte im RCS herstellt. Zunächst gibt es diese Objekte nicht als Grundtyp. Dehalb benutzen wir für Überschriften z.B. den Typ G_TEXT (G_STRING funktioniert in unserem Beispiel allerdings nicht, da Strings nicht im Replace-Modus gezeichnet werden). Jeder andere Typ funktioniert aber auch, da nur die y-Position des Objektes verändert wird. Die Zeichenkette für die Überschrift legen wir in eine Box direkt an den oberen Rand. Beim Text lassen wir aus ästhetischen Gründen links und rechts eine Leerstelle frei (siehe Abb. 2.17). Nun müssen wir nur noch den erweiterten Typ 32 (Hex 0x20) im oberen Byte von ob_type eintragen. Auf dem ATARI ST kann dies durch direkte Eingabe gemacht werden. Auf dem PC mit dem RCS geht es erst ab der Version 2.2. Für alle diejenigen, die nicht die Möglichkeit haben, packen wir auf die Diskette die Resoure-Datei aus obiger Abbildung. Man kann sich dann die gewünschten neuen Objekte einfach herauskopieren.

Um eine neue Checkbox zu erstellen, nehmen wir einfach ein Objekt vom Typ G_BOX (weiße Box) und positionieren es an die gewünschte Stelle. Wir stellen die Rahmenfarbe und das Füllmuster auf schwarz oder eine andere beliebige Farbe. Der erweiterte Typ wird auf 64 (Hex 0x40) gestellt. Das Bit SELECTABLE muß natürlich gesetzt werden. Die Box kann beliebig vergrößert werden.

Beim neuen Radio-Knopf benutzen wir die gleiche Methode wie bei der Checkbox. Hier stellen wir den erweiterten Typ auf 96 (Hex 0x60). Radio-Knöpfe haben immer eine Größe von 12x12 Pixel, da in Wirklichkeit ein Piktogramm gezeichnet wird. Deshalb nützt es nichts, wenn die Box größer gemacht wird.

Wie geht das Benutzen von eigenen Objekttypen im AES jetzt also vor sich? Dazu sehen wir uns die beiden Strukturen (AES.H) USERBLK und PARMBLK an.

```
typedef struct user_blk
{
  WORD (*ub_code) (); /* pointer to drawing function    */
  LONG ub_parm;       /* parameter for drawing function */
```

```
} USERBLK;

typedef struct parm_blk
{
  OBJECT *pb_tree;
  WORD   pb_obj;
  WORD   pb_prevstate;
  WORD   pb_currstate;
  WORD   pb_x, pb_y, pb_w, pb_h;
  WORD   pb_xc, pb_yc, pb_wc, pb_hc;
  LONG   pb_parm;
} PARMBLK;
```

Zur Verdeutlichung sehen wir uns das Beispiel Checkbox an. Die Chekbox wird im RCS als Objekt vom Typ G_BOX eingetragen. Damit das AES aber erkennt, daß nicht die normale Box, sondern der neue Typ gezeichnet werden soll, muß man den Objekttyp zur Laufzeit ändern. Dazu legen wir uns für jeden neuen Objekttyp eine Variable vom Typ USERBLK an.

```
LOCAL USERBLK check_blk;
```

Bei allen Objekten vom erweiterten Typ DCHECKBOX wird nach dem Laden der Resource-Datei folgendes gemacht: Zunächst wird die Zeichenfunktion (die Adresse der Funktion), die das neue Objekt darstellen soll, in das Feld ub_code eingetragen. In unserem Fall heißt die Funktion draw_checkbox.

```
check_blk.ub_code = draw_checkbox;
```

Sei „ob" ein Zeiger auf das zu ändernde Objekt mit Namen SBESTFIT (aus dem RCS), also

```
ob = &dialog [SBESTFIT];
```

Dann müssen wir noch folgende Aktionen vornehmen: In das Feld ub_parm wird der aktuelle Wert des ob_spec eingetragen, damit dieser nicht verloren geht. Anschließend wird der Typ unseres neuen Objekts auf G_USERDEF gesetzt. Schließlich müssen wir dem AES noch mitteilen, wie es die neue Zeichenfunktion finden soll. Dazu muß der ob_spec-Wert auf die USERBLK-Struktur (in unserem Beispiel check_blk) zeigen.

```
check_blk.ub_parm  = ob->ob_spec;
ob->ob_type        = G_USERDEF;
ob->ob_spec        = (LONG)&check_blk;
```

Jetzt sind wir im Prinzip fertig, wenn die neue Zeichenfunktion erstellt ist. Die neue Zeichenfunktion wird vom AES angesprungen, wenn das Objekt gezeichnet oder geändert werden soll (z.B. anderer Objektstatus). Die Zeichenfunktion (z.B. draw_checkbox)

wird vom AES mit einem einzigen Parameter versorgt, einem Zeiger auf die PARMBLK-Struktur (siehe AES.H). Mit Hilfe dieser Struktur erhält die Zeichenfunktion alle notwendigen Informationen über das Objekt, wie die Objektnummer, den Baum, an dem das Objekt hängt, den Status des Objekts, die Bildschirmkoordinaten usw. Die nachfolgende Zeichenfunktion draw_checkbox zeigt, wie man eigene erweiterte Objekte in Dialogboxen zeichnet. Achtung! In der Zeichenfunktion dürfen keinerlei AES-Aufrufe getätigt werden!

```
#if MSDOS
LOCAL WORD CDECL draw_checkbox ()
{
  PARMBLK *pb;
#else
LOCAL WORD CDECL draw_checkbox (pb)
PARMBLK *pb;

{#endif

  LONG    ob_spec;
  WORD    ob_x, ob_y, ob_width, ob_height;
  BOOLEAN selected, changed;
  WORD    pxy [10];

#if MSDOS
  pb = fardr_start ();
#endif

  ob_spec   = pb->pb_parm;
  ob_x      = pb->pb_x;
  ob_y      = pb->pb_y;
  ob_width  = pb->pb_w;
  ob_height = pb->pb_h;
  selected  = pb->pb_currstate & SELECTED;
  changed   = (pb->pb_currstate ^ pb->pb_prevstate) & SELECTED;

  set_clip (pb->pb_x, pb->pb_y, pb->pb_w, pb->pb_h);

  vsl_type (vdi_handle, FIS_SOLID);
  vsl_ends (vdi_handle, SQUARED, SQUARED);
  vsl_width (vdi_handle, 1);
  vsl_color (vdi_handle, obinfo.bdc);
  vswr_mode (vdi_handle, MD_REPLACE);

  if (! changed) /* war ein objc_draw-Aufruf */
  {
```

```
    pxy [0] = ob_x;
    pxy [1] = ob_y;
    pxy [2] = ob_x + ob_width - 1;
    pxy [3] = ob_y;
    pxy [4] = pxy [2];
    pxy [5] = ob_y + ob_height - 1;
    pxy [6] = ob_x;
    pxy [7] = pxy [5];
    pxy [8] = ob_x;
    pxy [9] = ob_y; /* Rahmen der Box zeichnen */

    v_pline (vdi_handle, 5, pxy);
  } /* if */

  if (selected) /* war objc_change-Aufruf */
    vsl_color (vdi_handle, BLACK);
  else
    vsl_color (vdi_handle, WHITE);

  set_clip (ob_x + 1, ob_y + 1, ob_width - 2, ob_height - 2);

  pxy [0] = ob_x;
  pxy [1] = ob_y;
  pxy [2] = ob_x + ob_width - 1;
  pxy [3] = ob_y + ob_height - 1;
  v_pline (vdi_handle, 2, pxy);

  pxy [0] = ob_x + ob_width - 1;
  pxy [1] = ob_y;
  pxy [2] = ob_x;
  pxy [3] = ob_y + ob_height - 1;
  v_pline (vdi_handle, 2, pxy);

#if MSDOS
  fardr_end ();
#endif

  return (pb->pb_currstate & ~SELECTED);
} /* draw_checkbox */
```

Der Anfang der Funktion ist vielleicht für manchen etwas unübersichtlich, konnte aber nicht anders gemacht werden. Der Zeiger auf den PARMBLK wird vom AES auf dem ATARI und unter X/GEM auf FLEXOS korrekt auf dem Stack übergeben, wie dies im AES-Handbuch in Kapitel 6.8 angegeben ist. Unter MS-DOS wird der Zeiger in Registern übergeben. Aus diesem Grund müssen die Funktionen fardr_start und fardr_end am Anfang und am Ende der Zeichenfunktion aufgerufen werden. Das Wort CDECL

bedeutet für den Compiler, daß die Parameter auf dem Stack übergeben werden sollen. Für Turbo-C auf dem ATARI ST ist diese Angabe unerläßlich.

Wichtig zu erwähnen ist auch die Variable changed. Ist changed FALSE, so wurde das Objekt (zum erstenmal) gezeichnet. In unserem Beispiel wird der Rand der Box gezeichnet (if ! changed ...). Anschließend wird das Innere der Box (das Kreuz) entweder in der Hintergrundfarbe Schwarz oder in der Farbe Weiß gezeichnet. Im ersten Fall ist das Objekt selektiert, im zweiten Fall muß das Innere mit weiß gelöscht werden.

Als Funktionswert sollte der augenblickliche Status des Objekts zurückgegeben werden. Das AES unternimmt dann noch einige Aktionen je nach Funktionswert. Ist der Status z.B. DISABLED, so wird das Objekt grau (nicht anwählbar) angezeigt. Alle Stati bis auf SELECTED können zurückgegeben werden. Wird der Status SELECTED zurückgegeben, so wird das gesamte Objekt auf dem Bildschirm invertiert, was wir bei unserer Checkbox nicht haben möchten. Deshalb blenden wir das SELECTED-Bit aus dem Funktionswert aus (pb_currstate & ~SELECTED).

Im Beispielprogramm SHOWGEM werden die drei neuen Objekttypen unterstützt. Dort sieht man auch den Zusammenhang zwischen der Initialisierung der neuen Objekte und den AES-Strukturen.

Bei der Beschreibung der nachfolgenden Funktionen wird davon ausgegangen, daß ein Objektbaum mit dem RCS erstellt wurde. Der Objektbaum ist als

```
OBJECT *tree;
```

definiert. Die Variable tree ist also ein Zeiger auf einzelne Objekte (bzw. ein Feld von Objekten, was in der Sprache C semantisch das Gleiche ist). In Pascal würde man

```
OBJECT tree [NUM_OBJ];
```

schreiben, wenn man die Anzahl der Objekte vorher wüßte.

- OBJC_ADD
- OBJC_DELETE

Diese beiden Funktionen wurden in anderen Büchern nie genau beschrieben. Es ist auch verständlich, da man das Aufbauen von Objektbäumen interaktiv mit Hilfe des RCS vornehmen kann. Nur so viel sei gesagt: Möchte man selbst Objektbäume aufbauen, so muß man den Platz für die Objekte bereits reserviert haben, das Feld mit den Objekten liegt also bereits vor. Mit Hilfe von objc_add kann dann eine neue Hierarchie (Vater-Sohn-Beziehung) bestimmt werden.

Mit Hilfe von objc_delete kann ein Objekt aus einem bestehenden Objektbaum gelöscht werden. Dabei wird lediglich das zu löschende Objekt aus dem Baum abgehängt, es bleibt quasi als Leiche im Feld tree stehen.

– OBJC_DRAW

Diese Funktion wird sehr häufig verwendet. Damit kann ein einzelnes Objekt oder ein ganzer Objektbaum gezeichnet werden. Die Tiefe der Hierarchie kann ebenfalls angegeben werden. Meist wird bis zur Hierarchie 8 gezeichnet, da es keine Objektbäume gibt, die mehr Stufen haben. Der Bereich auf dem Bildschirm (Clipping-Rechteck) wird ebenfalls angegeben. Beispiel:

```
objc_draw (tree, ROOT, MAX_DEPTH, xdial, ydial, wdial, hdial);
```

– OBJC_FIND

Hiermit kann untersucht werden, ob sich ein Objekt unter dem Mauszeiger befindet. Man erhält die Objektnummer. Hat man z.B. einen eigenen Desktop (siehe Beispielprogramm SCRAP), so kann man herausfinden, ob der Benutzer auf ein Piktogramm geklickt hat. Beispiel:

```
obj = objc_find (desktop, ROOT, MAX_DEPTH, x_mouse, y_mouse);
```

– OBJC_OFFSET

Da die x- und y-Koordinaten von Objekten in Objektbäumen (oder Dialogboxen) immer relativ zum Vaterobjekt angegeben sind, kann es mitunter schwierig zu bestimmen sein, wo genau sich ein Objekt auf dem Schirm befindet. Die Funktion objc_offset berechnet zum angegebenen Objekt die absoluten x- und y-Bildschirmkoordinaten. Beispiel: Berechnen der x- und y-Koordinate des Knopfes CANCEL im Programm SHOWGEM:

```
WORD x, y;

objc_offset (dialog, DCANCEL, &x, &y);
```

– OBJC_ORDER

Auch diese Funktion wird fast nie benutzt. Mit ihr kann man die Hierarchieordnung von Objekten in einem Baum zur Laufzeit verändern. Im allgemeinen behalten Objektbäume ihre Ordnung während des gesamten Programmlaufs bei.

– OBJC_EDIT

Diese Funktion wird ebenfalls nur selten benutzt. Mit ihr können Texte eines bestimmten Objekttyps editiert werden. In allen uns bekannten Büchern, außer „Softwareentwicklung auf dem ATARI ST“ ist der Objekttyp falsch angegeben. Sogar in den Original-

GEM-Unterlagen von Digital Research. Dort steht, daß das Objekt vom Typ G_TEXT oder G_BOXTEXT sein muß. In Wahrheit können aber nur Objekte vom Typ G_FTEXT und G_FBOXTEXT editiert werden.

Normalerweise benutzt man die Funktion form_do, um die komplette Abhandlung einer Dialogbox durch das AES vornehmen zu lassen. Die Felder, in die Texte eingetragen werden können, werden vom AES gehandhabt. Nur wenn man seine eigene form_do-Routine schreiben möchte, benötigt man die objc_edit-Funktion.

– OBJC_CHANGE

Mit Hilfe der Funktion objc_change kann der Status (ob_state) eines Objekts geändert und das Objekt auf Wunsch neu gezeichnet werden. Im Beispielprogramm SCRAP wird diese Funktion bei der Verwaltung des eigenen Desktop benutzt. Zieht ein Benutzer ein Piktogramm auf ein anderes, so wird es invertiert dargestellt. Der Status des Objekts, auf dem der Mauszeiger sitzt, muß also auf SELECTED gesetzt und das Objekt neu gezeichnet werden.

Beispiel: Der Status des Druckerpiktogrammes (das SELECTED-Bit) soll invertiert und das Objekt neu gezeichnet werden.

```
objc_change (desktop, IPRINTER, 0, desk.x, desk,y, desk.w, desk.h,
             desktop [IPRINTER].ob_state ^ SELECTED, TRUE);
```

5. FORM-LIBRARY

| | |
|---|---|
| 50 – form_do | Dialogbox abhandeln |
| 51 – form_dial | Bildschirmbereiche reservieren |
| 52 – form_alert | Alert-Box anzeigen |
| 53 – form_error | DOS-Fehlermeldung anzeigen |
| 54 – form_center | Dialogbox zentrieren |
| 55 – form_button | Mausknopf-Eingaben abhandeln |
| 56 – form_keybd | Tastatureingaben abhandeln |

Ein Formular (form) ist ein mit dem RCS erzeugter Objektbaum, zu dem man auch häufig Dialogbox sagt. Um die Interaktion von Dialogboxen mit dem Benutzer zu erleichtern, wird die Form-Library vom AES zur Verfügung gestellt.

Mit Hilfe eines einzigen Aufrufs (form_do) kann man den gesamten Dialog mit dem Benutzer vom AES abhandeln lassen. Das AES speichert dann die Zustände der Dialogbehandlung im Objektbaum der Dialogbox ab. Wählt der Benutzer z.B. Knöpfe an, die dann selektiert sind, so wird im Status des Objekts das SELECTED-Bit gesetzt. Die Applikation kann diese Informationen nach Abhandlung des Dialogs aus dem Objektbaum

herauslesen, um entsprechend zu agieren. Auch Texte, die vom Benutzer in Textfeldern editiert werden können, können nach Abhandlung des Dialogs ausgelesen werden.

Der Vorteil liegt klar auf der Hand. Zum einen muß sich der Programmierer nicht um das Handling der Dialogboxen kümmern, und zum anderen ist die Bedienung jeder Dialogbox für den Benutzer gleich und zwar über jedes GEM-Programm hinweg. Der Benutzer muß also nicht umlernen, wenn er eine Dialogbox von GEMDRAW oder WORDPLUS ausfüllen muß.

Eine Alertbox (form_alert) ist nichts anderes als eine kleine Dialogbox, in denen meist Fehlermeldungen angezeigt werden, die zu einer sofortigen Handlung zwingen sollen.

**Abb. 2.19:** Eine Beispielalertbox

Eine Alertbox kann bis zu 3 Knöpfe und 5 Zeilen Text beinhalten. Ein Piktogramm auf der linken Seite der Box zeigt die Art des Fehlers an (Notiz, Warten, Stop). Alert-Boxen können durch eine Zeichenkette beschrieben werden, die folgenden Aufbau besitzen muß:
[Piktogramm-Nummer][Text][Knöpfe]

Die Piktogrammnummer gibt an, welche Art Piktogramm erscheinen soll. Dabei gilt:

0 = kein Piktogramm
1 = Notiz-Piktogramm (Hinweis auf mögliche Fehler)
2 = Warte-Piktogramm (meist um eine Aktion zu wiederholen)
3 = Stop-Piktogramm (fatale Fehler, wie z.B. Diskette defekt)

Der Text hat folgenden Aufbau:

Textzeile 1|Textzeile 2| usw.

Die einzelnen Textzeilen werden also mit dem Pipe-Symbol '|' (logisches „oder" in C) getrennt. Das gleiche Format gilt auch für die Knöpfe.

Text von Knopf 1|Text von Knopf 2|Text von Knopf 3

Der gesamte Text aus Abbildung 2.19 muß dann wie folgt aufgebaut sein:

BYTE *alert_text;

alert_text = "[2][Laufwerk A: antwortet nicht.|Bitte Laufwerk überprüfen oder|Diskette einlegen][ABBRUCH|WEITER]";

Error-Boxen (form_error) sind spezielle Alert-Boxen, die nur Betriebssystemfehler anzeigen. Der Text wird von der Form-Library erzeugt. Man übergibt lediglich die Fehlernummer des Betriebssystems.

– FORM_DO

Wie oben erwähnt, wird durch form_do die gesamte Interaktion einer Dialogbox mit dem Benutzer abgehandelt. Man übergibt den Objektbaum und das erste Objekt, das editiert werden soll (auf das sich der Cursor stellen soll). Gibt es keine editierbaren Textfelder, so muß eine 0 übergeben werden. Man bekommt die Objektnummer zurückgeliefert, mit der der Benutzer das Formular verlassen hat.

Achtung! Wird ein Objekt, bei dem das TOUCHEXIT-Flag gesetzt ist, mit einem Doppelklick verlassen, so ist bei der zurückgelieferten Objektnummer das höchste Bit gesetzt. Wird kein Doppelklick benötigt, sollte man dieses Bit ausblenden.

Beispiel:

```
exit_obj = form_do (dialog, 0) & 0x7FFF;
if (exit_obj == DOK) /* Benutzer hat OK gedrückt */
{
  usw.
} /* if */
```

– FORM_DIAL

Hiermit können bis zu vier Aktionen durchgeführt werden:

- Bildschirmbereich als belegt kennzeichnen
- Zeichnen eines sich ausdehnenden Rechtecks
- Zeichnen eines schrumpfenden Rechtecks
- Bildschirmbereich wieder freigeben

Die beiden mittleren Optionen sind nur noch bei GEM 1.X gültig und haben ab GEM 2.X keine Bedeutung mehr.

Soll eine Dialogbox auf den Bildschirm gezeichnet werden, so muß die Applikation dem AES mitteilen, welchen Bereich des Bildschirms die Dialogbox benötigt. Dies wird mit

dem Aufruf form_dial und der Konstanten FMD_START erledigt. Außerdem übergibt man den Bildschirmbereich der Dialogbox.

Wird der Dialog beendet, muß das AES aus dem angegebenen Bereich Meldungen an die verschiedenen Applikationen senden (WM_REDRAW), damit diese den Inhalt ihrer Fenster neu zeichnen können. Nach Beendigung des Dialoges wird dies mit form_dial und der Konstanten FMD_FINISH erledigt.

Im GEM 1.X können außerdem die Konstanten FMD_GROW und FMD_SHRINK benutzt werden, wenn sich ausdehnende oder schrumpfende Rechtecke gezeichnet werden sollen.

– FORM_ALERT

Zeigt eine Alertbox an. Man übergibt den Default-Knopf, den der Benutzer mit der Return-Taste auswählen kann und erhält den Knopf, den der Benutzer gedrückt hat.

Beispiel mit alert_text von oben, wobei der Knopf „ABBRUCH“ Nummer 1 ist:

```
WORD button;
button = form_alert (2, alert_text);
if (button == 1) /* ABBRUCH gewählt */
{
} /* if */
```

– FORM_ERROR

Möchte man Fehler des Betriebssystems anzeigen wie etwa „Datei nicht gefunden“ etc., so kann man form_error benutzen. Man übergibt die Nummer des Betriebssystem-Fehlers und erhält den Knopf, der gedrückt wurde.

– FORM_CENTER

Mit der Funktion form_center werden die Bildschirmkoordinaten der Wurzel eines Objektbaumes so modifiziert, daß der Objektbaum (die Dialogbox) in der Mitte des Bildschirms erscheint. Eine Anwendung kann man aus der Funktion hndl_dial im Programm SHOWGEM sehen.

Ab GEM/3 wird eine Dialogbox in y-Richtung nicht mehr auf dem ganzen Bildschirm zentriert. Das liegt im Aufkommen der Großbildschirme begründet. Würde eine kleine Dialogbox immer in der Mitte des Bildschirms erscheinen, so müßte der Benutzer mit der Maus einen erheblichen Weg zurücklegen, bis er von der Menüleiste zur Dialogbox gefahren ist.

- FORM_BUTTON
- FORM_KEYBD

Diese beiden Funktionen werden intern von der form_do-Routine benutzt, um Tastatur- und Mausaktionen beim Bearbeiten einer Dialogbox abzuhandeln. Sie werden höchstens benutzt, um eigene form_do-Routinen zu schreiben.

## 6. GRAPHICS-LIBRARY

| | | |
|---|---|---|
| 70 | graf_rubbox | Zieht ein Gummiband |
| 71 | graf_dragbox | Box mit Maus verschieben |
| 72 | graf_mbox | Zeichnet eine sich bewegende Box |
| 73 | graf_growbox | Zeichnet ein sich ausdehnendes Rechteck |
| 74 | graf_shrinkbox | Zeichnet ein schrumpfendes Rechteck |
| 75 | graf_watchbox | Überwacht eine Box |
| 76 | graf_slidebox | Schieber zeichnen |
| 77 | graf_handle | VDI-Handle der Arbeitsstation holen |
| 78 | graf_mouse | Mausform ändern |
| 79 | graf_mkstate | Maus-/Tastaturinfo holen |

Wenn man den GEM-Desktop zum Kopieren von Dateien benutzt, so kann man ein Gummiband aufziehen, um mehrere Dateien einzurahmen. Man wird es auch des öfteren in einer Applikation damit zu tun haben, rechteckige Rahmen zu manipulieren. Auch die Abfrage der Mausposition, die ja nicht mit VDI-Befehlen durchgeführt werden kann, wird mit Funktionen aus der Graphics-Library realisiert.

### - GRAF_RUBBOX

In vielen Bindings verschiedener C-Compiler auf dem ATARI ST ist der Name der Funktion falsch angegeben. Dort erscheint dann graf_rubberbox. Diese Funktion existiert aber nicht, sie wird von Digital Research in den Original-GEM-Bindings mit graf_rubbox angegeben. Sie können auf jeden Fall den Originalnamen benutzen, wenn Sie die Datei PORTAB.H inkludieren. Sie merzt solche Fehler aus.

Mit Hilfe dieser Funktion kann mit der Maus ein Gummiband aufgezogen werden. Interessant ist, daß bei Angabe der minimalen Breite und Höhe auch negative Werte zugelassen sind. Das Gummiband läßt sich dann auch von rechts unten nach links oben ziehen.

### - GRAF_DRAGBOX

Hiermit kann eine quadratische Box innerhalb eines angegebenen Rechtecks verschoben werden. Man übergibt die Position, Breite und Höhe der Box, sowie den Bereich, in dem

die Box verschoben werden kann. Das AES übernimmt dann beim Drücken der Maustaste automatisch das Verschieben der Box und kehrt erst dann zurück, wenn der Benutzer den Mausknopf wieder losgelassen hat. Die Position der verschobenen Box wird zurückgegeben.

– GRAF_MBOX

Auch hier ist der Name in vielen Bindings mit graf_movebox angegeben. Der Originalname ist aber graf_mbox, der mit Hilfe von PORTAB.H immer benutzt werden kann (graf_movebox und graf_mouse sind in den ersten 7 Buchstaben identisch). Eine sich bewegende Box von einer Position zu einer anderen wird gezeichnet. Man wird dies dort benutzen, wo das Verschieben eines Piktogrammes nicht erlaubt ist und es an seine ursprüngliche Stelle zurückpositioniert wird (z.B. Drucker auf Papierkorb legen).

– GRAF_GROWBOX
– GRAF_SHRINKBOX

Diese beiden Funktionen werden ab GEM 2.X nicht mehr unterstützt. Sie dienen dazu, sich vergrößernde und verkleinernde Rechtecke zu zeichnen. Man benutzt sie z.B., wenn sich ein Fenster aus dem Öffnen eines Piktogrammes ergibt, damit der Benutzer weiß, von welchem Objekt aus das Fenster geöffnet wurde. Auf dem Macintosh werden diese Boxen immer benutzt.

Um sie dennoch zu zeichnen, existiert die Extended-Graphics-Library. Wie man mit ihr die fehlenden Boxen auch ab GEM 2.X wieder herbeizaubern kann, wird im Beispielprogramm SCRAP gezeigt.

– GRAF_WATCHBOX

Diese Funktion wird von einer Applikation eigentlich nie benutzt. Die FORM-Library benötigt sie, wenn man mit der Maus in einer Dialogbox auf ein Exit-Objekt klickt und die Maustaste noch gedrückt hält. In diesem Falle kann der Status eines Objektes (z.B. der OK-Knopf) invertiert werden, wenn der Benutzer mit gedrückter Maustaste in die Box fährt und sie wieder verläßt.

– GRAF_SLIDEBOX

Mit der Funktion graf_slidebox kann ein horizontaler oder vertikaler Schieber erzeugt werden. Die WINDOWS-Library benutzt diesen Aufruf für die horizontalen bzw. vertikalen Schieber. Auch die FILE-SELECTOR-LIBRARY benutzt diesen Aufruf, wenn in der Dateiauswahlbox so viele Dateien angezeigt werden, daß man scrollen kann.

– GRAF_HANDLE

Wie schon in Kapitel 2.3 erklärt, öffnet das AES den Bildschirm als physikalische VDI-Arbeitsstation. Da unsere Applikation den Bildschirm auch benötigt und wir diesen nur noch virtuell öffnen dürfen, benötigen wir die physikalische Kennung (handle) des Bildschirms bei der Funktion v_opnvwk. Diese Kennung erhalten wir mit der Funktion graf_handle.

Außer der VDI-Kennung bekommen wir noch 4 weitere Größen zurückgeliefert, die enorm wichtig sind. Die ersten beiden geben die Breite und Höhe einer Zeichenzelle im Systemzeichensatz an. Der Systemzeichensatz wird in Menüs und Dialogboxen benutzt. Die beiden letzten Größen geben die Breite und Höhe einer Box an, die bei Fensterattributen benutzt werden. Fensterattribute sind z.B. die Pfeile, Schließbox, Vergrößerungsbox usw.

Beispiel:

```
WORD phys_handle;
WORD gl_wbox, gl_hbox, gl_wattr, gl_hattr;

phys_handle = graf_handle (&gl_wbox, &gl_hbox, &gl_wattr, &gl_hattr);
```

– GRAF_MOUSE

Mit dieser Funktion kann das Aussehen der Maus verändert werden. Die Mausform kann von einem Pfeil z.B. zu einer Sanduhr (MS-DOS) oder einer Biene (ATARI ST) geändert werden, um anzuzeigen, daß der Rechner gerade arbeitet. Auch selbstdefinierte Mausformen können erstellt werden. Im Beispielprogramm SCRAP wird davon Gebrauch gemacht.

– GRAF_MKSTATE

Da bei AES-Applikationen keine VDI-Eingabefunktionen aufgerufen werden dürfen, um z.B. die Maus und Tastatur abzufragen, bietet das AES die Funktion graf_mkstate (Maus-Keyboard-Status) an. Dabei werden die Mauskoordinaten, die Mausknöpfe, die linke und rechte Shift-Taste und die Control- und Alternate-Taste zurückgeliefert. Ein Bit sagt jeweils aus, ob die Taste gedrückt (1) oder losgelassen ist (0).

7. SCRAP-LIBRARY

| | |
|---|---|
| 80 – scrp_read | Klemmbrett-Directory lesen |
| 81 – scrp_write | Klemmbrett-Directory schreiben |
| 82 – scrp_clear | Klemmbrett-Directory löschen |

Wer schon einmal mit einem Macintosh gearbeitet hat, der weiß, daß es möglich ist, Daten zwischen beliebigen Applikationen auszutauschen. So kann man eine Zeichnung mit MacPaint erstellen und sie anschließend z.B. in einem Desktop-Publishing-Programm einbinden. Dasselbe gilt auch für objektorientierte Vektorzeichnungen mit MacDraw.

Auf dem ATARI ST z.B. ist es bisher kaum möglich, Daten aus verschiedenen GEM-Applikationen auszutauschen. Digital Research hatte aber schon immer im GEM diese Möglichkeiten vorgesehen. Bisher sind sie nur noch nicht beschrieben bzw. benutzt worden.

Um Daten von einer Applikation zu einer anderen Applikation auszutauschen, soll es folgende Funktionen geben: Eine Zwischenablage (Klemmbrett, Clipboard) soll die Daten, sogenannte Scraps, aufnehmen können. Daten können von einer Applikation aus entweder ausgeschnitten (Cut), kopiert (Copy) oder eingefügt (Paste) werden.

Beim Ausschneiden werden die selektierten Daten aus der laufenden Applikation gelöscht und in das Systemklemmbrett kopiert. Beim Kopieren bleiben die Daten in der laufenden Applikation erhalten und werden auf das Klemmbrett kopiert. Beim Einfügen schließlich werden die Daten aus dem Systemklemmbrett entnommen und in die neue Applikation hineinkopiert.

Das Systemklemmbrett ist ein Verzeichnis (Ordner, Subdirectory) auf einer Diskette oder Festplatte. In älteren GEM-Versionen (bis GEM 2.X) hieß dieses Verzeichnis GEMSCRAP. Ab GEM 3.X heißt es CLIPBRD und befindet sich im Verzeichnis GEMAPPS.

Für den ATARI ST gibt es zur Zeit genau ein einziges Programm, das die Zwischenablage konsequent nutzt: Die neueste Version von WORDPLUS. Dazu legt es sich im Wurzelverzeichnis den Ordner CLIPBRD an, wenn er noch nicht existiert. Wir vereinbaren an dieser Stelle also: Das Verzeichnis für die Zwischenablage auf dem ATARI ST heißt CLIPBRD und befindet sich im Wurzelverzeichnis der Diskette oder der Festplatte. Jedes Programm, das in Zukunft diese Zwischenablage nutzt, hat diesen Ordner bei Nichtvorhandensein anzulegen oder ihn zu benutzen, wenn er schon da ist. Das Programm SCRAP übernimmt die Verwaltung dieses Systemklemmbrettes und wird in Kapitel 6 genauestens erläutert.

Für MS-DOS gilt ab GEM Version 3.X, daß sich das Systemklemmbrett im Ordner \GEMAPPS\CLIPBRD befindet.

Um verschiedenartige Daten auszutauschen, unterstützt das AES folgende Dateitypen:

a) CSV = Comma separated values
b) TXT = ASCII-Dateien
c) DIF = Daten von Tabellenkalkulationen
d) GEM = Metadateien
e) IMG = Bit-Image-Dateien
f) DCA = IBM document contents architecture
g) USR = benutzerdefiniertes Format

Alle Dateien beginnen mit dem Dateinamen SCRAP und haben einen der o.g. Suffixe. Eine Metadatei erscheint also als SCRAP.GEM, wenn diese aus einem Zeichenprogramm ins Systemklemmbrett geschrieben wird. Aus dieser Forderung ergibt sich, daß das Klemmbrett nur eine Stufe tief ist. Kopiert man also zwei GEM-Dateien hinein, so wird die erste Datei automatisch überschrieben.

Um Daten verschiedenen Programmen zugänglich zu machen, darf man die ausgeschnittenen oder kopierten Daten auch in verschiedenen Formaten abspeichern. Ein WORDPLUS-Textstück könnte als SCRAP.TXT und als SCRAP.1WP im Verzeichnis CLIPBRD erscheinen. Ersteres ist das Textstück als reine ASCII-Datei, zweites ist das gleiche Textstück mit den Formatanweisungen für WORDPLUS. So kann wenigstens die Textdatei in andere Programme übernommen werden, während die 1WP-Datei in die Programme übernommen werden kann, die das Format von WORDPLUS verstehen.

Weitere Informationen zum Systemklemmbrett befinden sich in Kapitel 6, wo die Arbeitsweise genau beschrieben wird.

– SCRP_READ

Mit dieser Funktion kann man den Pfad lesen, wo sich das Systemklemmbrett befindet. Außerdem bekommt man einen Bitvektor zurück, der das Vorhandensein der Dateien „SCRAP.*“ prüft. Ist kein Systemklemmbrett vorhanden, erhält man den Wert – 1. Für andere Werte gelten folgende Bitbelegungen:

Bit 0: SCRAP.CSV
Bit 1: SCRAP.TXT
Bit 2: SCRAP.GEM
Bit 3: SCRAP.IMG
Bit 4: SCRAP.DCA
Bit 15: SCRAP.USR

Alle anderen Bits sind für zukünftige Versionen reserviert.

Leider funktioniert scrp_read auf dem ATARI ST noch nicht. Zu diesem Zweck kann die Funktion scrap_read im Modul CLIPBRD des Programmes SCRAP aufgerufen werden.

– SCRP_WRITE

Hiermit wird das Systemklemmbrett eingerichtet, wenn es noch nicht vorhanden ist. Jede Applikation könnte sich also ein eigenes Klemmbrett einrichten. Wir raten dringend davon ab, da sonst das Chaos nicht mehr überschaubar wäre. Deshalb soll nur der o.g. Standardname eingetragen werden (für MSDOS: \GEMAPPS\CLIPBRD und für

ATARI ST: \CLIPBRD). Diese Funktion ist auf dem ATARI ST korrekt implementiert und kann direkt aufgerufen werden.

– SCRP_CLEAR

Alle Dateien mit dem Namen SCRAP und beliebigem Suffix werden im Klemmbrett-Verzeichnis gelöscht. Leider funktioniert der Aufruf scrp_clear auf dem ATARI ST nicht. Deshalb verwenden wir die Funktion scrap_clear aus dem Modul CLIPBRD des Programmes SCRAP.

8. FILE-SELECTOR-LIBRARY

| | | |
|---|---|---|
| 90 | – fsel_input | Dateiauswahlbox benutzen |
| 91 | – fsel_exinput | erweiterte Dateiauswahlbox benutzen |

– FSEL_INPUT

Zu dieser Library gibt es eigentlich nicht viel zu sagen. Jeder hat die Wirkung des Aufrufs fsel_input schon einmal selbst erlebt. Es erscheint die Dateiauswahlbox (Item selector, Objekt-Auswahl), mit der der Benutzer eine Datei anwählen kann. Startet man z.B. WORDPLUS, so erscheint sofort diese Box, um den Namen der Datei, die man editieren möchte, angeben zu können.

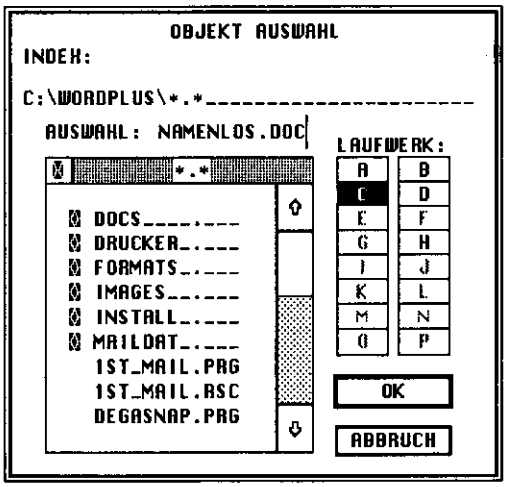



**Abb. 2.20:** Dateiauswahlbox ATARI ST (TOS 1.4)

Nach dem Aufruf bekommt man den Pfad und die Datei zurück, die man ausgewählt hat. Ebenfalls wird ein Wert zurückgeliefert, der aussagt, ob ein Benutzer den Knopf OK oder ABBRUCH gewählt hat.

– FSEL_EXINPUT

Die Funktion fsel_exinput ist nur auf dem ATARI ST ab TOS Version 1.4 verfügbar. Die erweiterte Dateiauswahlbox erlaubt der Applikation noch eine Zeichenkette mit höchstens 30 Zeichen zu übergeben, die dann am oberen Rand der Box angezeigt wird.

Beispiel: Zeige erweiterte Dateiauswahlbox mit Meldung „Text einlesen".

```
BYTE fs_iinpath [128];
BYTE fs_iinsel [12];
WORD fs_iexbutton;
BYTE fs_ilabel [31];

strcpy (fs_iinpath, "C:\\WORDPLUS\\*.DOC");
strcpy (fs_iinsel, "NEU.DOC");
strcpy (fs_ilabel, "Text einlesen");

fsel_exinput (fs_iinpath, fs_iinsel, &fs_iexbutton, fs_ilabel);
```

## 9. WINDOW-LIBRARY

| | |
|---|---|
| 100 – wind_create | Fenster kreieren |
| 101 – wind_open | Fenster öffnen |
| 102 – wind_close | Fenster schließen |
| 103 – wind_delete | Fenster löschen |
| 104 – wind_get | Fensterinformationen holen |
| 105 – wind_set | Fensteraufbau bestimmen |
| 106 – wind_find | Fenster unter Maus bestimmen |
| 107 – wind_update | Fenster restaurieren |
| 108 – wind_calc | Fenstergrößen holen |
| 109 – wind_new | Alle Fenster löschen |

In einem GEM-System können maximal 8 Fenster benutzt werden. 7 davon sind sogenannte Applikations-Fenster, das achte ist das Desktop-Fenster.

Das Desktop-Fenster besteht aus der Menüleiste und dem Bereich unter der Menüleiste, dem Arbeitsbereich. Dieses Fenster ist immer vorhanden und kann nicht gelöscht werden. Die meisten GEM-Programme benutzen dieses Fenster als Hintergrund für ihre Applika-

tion. Es wird vom AES verwaltet und besitzt die Kennung 0. In der Beispielapplikation SCRAP wird dieses Fenster mit der C-Konstanten DESK referiert (#define DESK 0).

Die anderen sieben Fenster werden von der Applikation benutzt, um irgendetwas darin anzuzeigen (Grafik, Texte usw.). Ein Fenster (siehe Abb. 2.15) besteht aus 2 Bereichen: Den Randbereichen (border area) wie Schließbox, Schieber, Infozeile und dem Arbeitsbereich (work area).

Alle Aktionen, die mit den Randbereichen eines Fensters durchgeführt werden, werden vom AES automatisch verwaltet. Verschiebt ein Benutzer ein Fenster, so überwacht das AES die Verschiebung und meldet der Applikation die neue Fensterposition über Nachrichten, in diesem Fall WM_MOVED (siehe Event-Library). Nachdem eine Meldung gesendet wurde, kann die Applikation entscheiden, ob sie die Aktion durchführen möchte. In unserem Beispiel kann die Applikation verhindern, daß das Fenster vom Benutzer verschoben wird.

Muß ein Fenster neu gezeichnet werden, so sendet das AES der Applikation die Nachricht WM_REDRAW. Die Applikation kann sich die Teile des Fensters holen, die neu gezeichnet werden müssen. Das AES verwaltet eine sogenannte Rechteckliste, in der zu jedem Fenster das Innere des Fensters in Rechtecke unterteilt wird. Liegt das Fenster oben, so gibt es genau ein Rechteck, nämlich das gesamte Fensterinnere. Liegt ein Fenster über einem anderen Fenster, so gibt es 2, 3 oder höchstens 4 Rechtecke, die neu gezeichnet werden müssen.

Die Fensterverwaltung ist im Prinzip in jeder Applikation gleich. Es ändert sich nur die Funktion, die das Fensterinnere zeichnet (z.B. die Funktion wi_draw, die in jedem Modul existiert, das mit Fenstern zu tun hat). Aus diesem Grund wurde der Windowmanager im Modul WINDOWS der Beispielapplikation SCRAP entwickelt. Er handelt fast alle Fensteraktionen automatisch ab. Man muß ihm nur noch die Zeichenfunktion und einige andere Funktionen übergeben. Der Manager überwacht die AES-Fensternachrichten und ruft die entsprechende Funktion in jedem Modul auf, das Fenster verwaltet. Die AES-Fensterfunktionen werden also ausschließlich vom Windowmanager benutzt und werden deshalb kurz beschrieben, falls man sich in fremde GEM-Applikationen einarbeiten muß.

– WIND_CREATE

Durch Aufruf von wind_create wird ein neues GEM-Fenster erzeugt. Man kann angeben, welche Randattribute (Schließbox, Vergrößerungsbox, Infozeile usw.) das Fenster bekommen kann (siehe AES.H). Außerdem gibt man an, welche Maximalgröße das Fenster jemals annehmen kann. Der Funktionswert ist die sogenannte Fensterkennung (window handle). Ist diese Kennung größer oder gleich 0, so konnte das Fenster erzeugt werden. Mit dieser Kennung werden dann alle weiteren Funktionen aus der Window-Library aufgerufen.

Ab GEM 2.X existiert eine neue Randkomponente:

```
#define HOTCLOSE 0x1000
```

Die „Hotclosebox" ist nichts anderes als eine Schließbox, bei der das TOUCHEXIT-Flag gesetzt ist. Klickt der Benutzer auf solch eine Schließbox und läßt den Mausknopf gedrückt, so wird vom AES ständig die Meldung WM_CLOSED gesendet. Der GEM Desktop ab Version 2.X benutzt diese Box, wenn man ein Fenster schließen möchte. Ist man z.B. in der 3. Ebene eines Inhaltsverzeichnisses, so muß man nicht 3-mal klicken, um auf die oberste Ebene zu gelangen. Man muß nur den Mausknopf auf der Schließbox gedrückt halten.

– WIND_OPEN

Ruft man wind_open auf, so erscheint das mit wind_create erzeugte Fenster auf dem Bildschirm. Die Anfangsposition und -größe wird mitübergeben.

Beispiel: Erzeugen eines Fensters mit Maximalgröße 200x200 und Öffnen des Fensters mit Größe 100x100.

```
WORD wh;

wh = wind_create (NAME+CLOSER, 0, 0, 200, 200);
if (wh >= 0)
{
  wind_open (wh, 0,0, 100, 100);
} /* if */
```

– WIND_CLOSE

Das Fenster mit der angegebenen Kennung kann durch wind_close geschlosssen werden. Das Fenster verschwindet vom Bildschirm. Die Kennung wird noch nicht freigegeben, womit das Fenster jederzeit wieder mit wind_open geöffnet werden kann.

– WIND_DELETE

Das Fenster mit der angegebenen Kennung wird komplett gelöscht und die Kennung wird freigegeben. Erst durch wind_create kann wieder ein neues Fenster beschafft werden. Der Vorteil von wind_delete ist, daß andere Prozesse (z.B. Accessories) die freien Kennungen weiter benutzen können, da ja nur sieben Fenster erzeugt werden können.

- WIND_GET
- WIND_SET

Informationen über ein Fenster können geholt oder gesetzt werden. Dazu gehören die augenblickliche Größe und Position, die Größe und Positionen der horizontalen und vertikalen Schieber, die Rechteckliste, der Name des Fensters usw. Dabei werden verschiedene Konstanten als Informationsunterscheidung übergeben (siehe AES.H, wind_get flags), und man erhält entsprechend 4 Rückgabewerte zu 16 Bit (WORD).

Beispiel: Name des Fensters setzen und Arbeitsbereich (work area) des Fensters holen.

```
WORD x, y, w, h;
BYTE *wind_name;

wind_name = '' Fenstername '';
wind_set (wh, WF_NAME, ADR (wind_name), 0, 0);
wind_get (wh, WF_WXYWH, &x, &y, &w, &h);
```

Bemerkung: Bei der Angabe des Fensternamens sollte man aus ästhetischen Gründen vor und hinter der Zeichenkette immer ein Leerzeichen lassen, da sonst das Muster im Fensternamen direkt am ersten und am letzten Buchstaben hängt.

- WIND_FIND

Möchte man zu einer gegebenen Mausposition wissen, welches Fenster sich unter der Maus befindet, so kann man wind_find aufrufen. Der Funktionswert ist die Fensterkennung.

- WIND_UPDATE

Mit der Funktion wind_update können 2 Zustände ein- oder ausgeschaltet werden. Soll das Innere eines Fensters neu gezeichnet werden, so muß dies dem AES mitgeteilt werden, damit es die Zeichenfunktion der Applikation nicht stört (BEG_UPDATE, END_UPDATE).

Der andere Zustand, der ein- oder ausgeschaltet werden kann, läßt die Applikation die gesamte Mauskontrolle übernehmen (BEG_MCTRL, END_MCTRL). Der Bildschirmmanager klinkt sich aus und die Applikation muß auf alle Mausereignisse selbst reagieren. Drop-Down-Menüs oder Randbereiche von Fenstern werden vom AES dann nicht mehr verwaltet. Dies ist nur dann sinnvoll, wenn das GEM-System die Applikation zu sehr einschränkt.

Beispiel: Fensterinhalt neu zeichnen.

```
wind_update (BEG_UPDATE);
/* Fensterinhalt zeichnen */
wind_update (END_UPDATE);
```

Anmerkung: Soll mit einem Accessory kommuniziert werden, so muß END_UPDATE eingeschaltet werden, damit die Nachrichten und Quittungen, die man sich gegenseitig sendet, vom AES übermittelt werden können.

– WIND_CALC

Durch die Funktion wind_calc kann der Arbeitsbereich oder Randbereich eines Fensters berechnet werden. Man übergibt die Position und Größe des Bereiches und erhält den jeweils anderen Bereich. Außerdem muß man die Attribute des Randbereiches angeben, die für das auszurechnende Fenster gelten sollen.

Beispiel: Aus dem Arbeitsbereich soll der Randbereich eines beliebigen Fensters berechnet werden.

```
WORD kind;
WORD x, y, w, h;
WORD rx, ry, rw, rh;

kind = NAME+CLOSER+FULLER; /* Fensterattribute */
x = x-Position des Arbeitsbereiches
y = y-Position des Arbeitsbereiches
w = Breite des Arbeitsbereiches
h = Höhe des Arbeitsbereiches
wind_calc (WC_BORDER, kind, x, y, w, h &rx, &ry, &rw, &rh);
```

Der Randbereich des Fensters befindet sich in den Variablen rx, ry, rw und rh.

– WIND_NEW

Diese Funktion ist nur auf dem ATARI ST ab TOS Version 1.4 verfügbar. Sie löscht und schließt alle Fenster, setzt die wind_update-Funktion zurück und setzt die Maus auf den Pfeil. Die Funktion wird z.B. aufgerufen, wenn ein Programm abgestürzt ist und man zum Desktop zurückkehrt. In älteren TOS-Versionen konnte man dann im Desktop meist keine Menüs mehr auswählen, da die abgestürzte Applikation sich meist in der Zeichenfunktion eines Fensters befand und dort die AES-Funktion wind_update (BEG_UPDATE) aufgerufen hatte.

10. RESOURCE-LIBRARY

| | |
|---|---|
| 110 – rsrc_load | Resource-Datei laden |
| 111 – rsrc_free | Speicher von Resource-Dateien freigeben |
| 112 – rsrc_gaddr | Adressen von Objekten holen |
| 113 – rsrc_saddr | Adressen von Objekten setzen |
| 114 – rsrc_obfix | x, y, Breite und Höhe konvertieren |

Ressourcen einer Applikation sind Fehlermeldungen, Dialogboxen und Menüzeilen. Sie werden getrennt vom Programmcode der Applikation aufbewahrt. Man erstellt sie mit einem Resource Construction Set und erhält eine Datei mit Suffix „.RSC". Die Vorteile von Resource-Dateien liegen klar auf der Hand: Möchte man z.B. die Meldungen einer Applikation in eine andere Sprache übersetzen, so muß nur die Resource-Datei übersetzt werden. Am Programmcode selbst muß nichts geändert werden.

Der nächste Vorteil liegt in der Unabhängigkeit von verschiedenen Bildschirmauflösungen und Grafikkarten. Eine Dialogbox, die 15 Textzeilen hoch ist und z.B. 10 Zeilen Text beinhaltet, wird in allen Bildschirmauflösungen die gleiche x-y-Position und Höhe haben. Dies liegt daran, daß in der Resource-Datei die x-y-Position und die Breite und Höhe einer Dialogbox in Vielfachen von Zeichensatzgrößen angegeben sind. Beim Laden der Resource-Datei werden diese Zeichengrößen mit Hilfe der Funktion rsrc_obfix mit der Breite und Höhe eines Zeichens aus dem Systemzeichensatz multipliziert.

Weitere Vorteile ergeben sich bei der Darstellung von Piktogrammen. In Resource-Dateien sind Piktogramme und Bit-Images im Standardformat abgelegt. Bei Benutzung von anderen Grafikkarten können diese Standardformate in das entsprechende gerätespezifische Format umgewandelt werden (siehe auch vr_trnfm).

Durch die Darstellung von Größen und Positionen in Zeichenkoordinaten und durch die Darstellung von Piktogrammen in Standardformaten sind Resource-Dateien portabel von einer Prozessorfamilie zu einer anderen (z.B. INTEL 80xxx zu Motorola 68xxx und umgekehrt).

– RSRC_LOAD
– RSRC_FREE

Das AES liest beim Aufruf von rsrc_load aus dem Kopf der Resource-Datei die Größe, alloziert dynamisch Speicher und lädt die Datei. Dann werden die x-y-Positionen sowie die Breiten und Höhen von Objekten an die Bildschirmauflösung angepaßt. Die Zeiger der Objekte, die z.B. auf TEDINFO's, ICONBLK's usw. zeigen, werden angepaßt und ein Feld aufgebaut, das Zeiger auf alle Objektbäume enthält. Die Adresse dieses Feldes wird in global [5] und global [6] abgelegt. Der Funktionswert ist FALSE, falls die Datei nicht geladen werden konnte.

Bemerkung: Ab GEM 2.X wird bei Objektbäumen, die als Wurzel eine Box besitzen (z.B. alle Dialogboxen), im Objektstatus das Flag SHADOWED gesetzt. Das bedeutet, daß alle Dialogboxen statt umrahmt mit einem Schatten gezeichnet werden. Dies sind noch Folgen des Rechtsstreits von Digital Research Incorporated mit Apple Computer. Apple hatte den Prozeß gewonnen, so daß DRI einige kosmetische Änderungen ab GEM 2.X vornehmen mußte. Z.B. bekam der GEM-Desktop nur noch 2 feste Fenster. Dialogboxen, bei denen im RCS das OUTLINED-Bit im Objektstatus gesetzt war, wurden modifiziert, so daß jetzt ein Schatten gezeichnet wird. Denn Dialogboxen mit umrahmter Box sehen genau so aus wie auf einem Apple-Macintosh. Man kann das SHADOWED-Bit aber nach dem rsrc_load zur Laufzeit wieder zurücksetzen und dafür das OUTLINED-Bit setzen (siehe RESOURCE.C).

Durch rsrc_free wird eine geladene Resource-Datei wieder freigegeben.

Beispiel:

```
BYTE *rsrc_name;
rsrc_name = "TEST.RSC";

if (! rsrc_load (rsc_name)) /* Resource-Datei nicht gefunden */
{
  strcpy (s, "[3][Resource-File|");
  strcat (s, rsc_name);
  strcat (s, "?][ EXIT ]");
  form_alert (1, s);
  return (FALSE);
} /* if */

/* Aktionen... */

rsrc_free ();
```

– RSRC_GADDR

Hiermit holt man sich Adressen von Objekten aus der geladenen Resource-Datei. Meist werden gleich nach dem Laden die Adressen aller Objektbäume geholt, damit man schnell auf diese zugreifen kann. Man gibt zuerst den Objekttyp, dann den Index des Objekts (aus der Datei TEST.H, die vom RCS erzeugt wurde) und die Variable an, in die das Ergebnis gespeichert werden soll. Welche Objekttypen geholt werden können, kann man in der Datei AES.H nachsehen.

Beispiel:

```
OBJECT *menu;
OBJECT *about;
```

```
/* Resource-Datei laden, s.o. ...*/

rsrc_gaddr (R_TREE, MENU,  &menu);
rsrc_gaddr (R_TREE, ABOUT, &about);
objc_draw (about, ROOT, MAX_DEPTH, xdial, ydial, wdial, hdial);
```

Das Programmstück lädt die Resource-Datei und holt sich von den beiden Objekbäumen die Adressen der Wurzel (R_TREE) in die Variablen menu und about. Anschließend wird die Dialogbox, die am Objektbaum about hängt, auf dem Bildschirm ausgegeben.

– RSRC_SADDR

Dies ist die Umkehrfunktion zu rsrc_gaddr. Allerdings können nur die beiden Objekttypen R_FRSTR und R_FRIMG gesetzt werden, was aber in der Praxis wohl nie vorkommt.

– RSRC_OBFIX

Diese Funktion wird von rsrc_load aufgerufen. Wie oben erwähnt, werden die x-y-Werte und die Breiten und Höhen aller Objekte mit Hilfe des Systemzeichensatzes in Pixelgrößen umgewandelt (siehe auch Modul RCM.C).

Beispiel: Konvertiere das Wurzelobjekt der Dialogbox about.

```
rsrc_obfix (about, 0);
```

## 11. SHELL-LIBRARY

| | |
|---|---|
| 120 – shel_read | Erkennt, wie eine Appl. gestartet wurde |
| 121 – shel_write | AES verlassen oder Applikation starten |
| 124 – shel_find | Datei suchen |
| 125 – shel_envrn | Liest die Umgebungsparameter |
| 126 – shel_rdef | Gibt die Default-Applikation zurück |
| 127 – shel_wdef | Setzt die Default-Applikation |

Die Shell-Library hat die Funktion, einer Applikation die Möglichkeit zu geben herauszufinden, wie sie gestartet wurde. Außerdem können Dateien gesucht und andere Programme gestartet werden.

Unter MS-DOS startet man das GEM-System normalerweise mit dem Kommando GEM. Nachdem das VDI und das AES geladen sind, startet der GEM-Desktop als sogenannte Default-Applikation. Startet man vom Desktop aus andere Programme, so kehrt man

automatisch zur Default-Applikation zurück. Dies kann aber verändert werden, falls man von einer Applikation A eine andere Applikation B aufrufen möchte und nach Beendigung der Applikation B wieder Applikation A und nicht den GEM-Desktop starten möchte.

– SHEL_READ

Durch einen Aufruf von shel_read bekommt man den Pfad und den Dateinamen der eigenen Applikation zurück. Außerdem bekommt man die Kommandozeile geliefert, die beim Aufruf eventuell übergeben wurde (z.B. einen Dateinamen, wenn die Applikation im GEM-Desktop installiert und das entsprechende Dokument angeklickt wurde).

Beispiel: Das Programm TEST.APP wurde vom Verzeichnis D:\GEMAPPS gestartet, indem die Datei CDS.DOC im Verzeichnis C:\WORDPLUS\DOCS angeklickt wurde. Vorher wurde die Datei mit „Anwendung anmelden..." vom GEM-Desktop aus installiert.

```
BYTE pcmd [128];    /* mindestens 128 Zeichen */
BYTE ptail [128];
shel_read (pcmd, ptail);
```

Die Zeichenkette pcmd enthält den Wert „D:\GEMAPPS\TEST.APP", die Zeichenkette ptail enthält den Wert „C:\WORDPLUS\DOCS\CDS.DOC".

– SHEL_WRITE

Hiermit können andere Programme aufgerufen werden. Dabei wird angegeben, ob das nachfolgende Programm sofort gestartet wird oder erst, wenn die laufende Applikation beendet ist. Außerdem wird mitgeteilt, ob es sich beim aufgerufenen Programm um eine GEM-Applikation handelt oder nicht. Der Name und die Kommandozeile (siehe shel_read) werden ebenfalls übergeben.

Die Übergabewerte können folgende Werte annehmen:

doex: 0 = AES verlassen, zurück zum Betriebssystem. Bei ATARI ST wird warm gebootet (Accessories werden neu geladen, z.B. beim Umschalten der Bildschirmauflösung).
1 = andere Applikation aufrufen

isgr: 0 = Textapplikation
1 = Grafikapplikation

isover: 0 = neue Applikation zusätzlich laden
1 = neue Applikation als Overlay (im Speicherbereich der alten Applikation) laden
2 = wie 1, aber zusätzlich das VDI aus dem Speicher entfernen

pcmd : Dateiname der neuen Applikation
ptail : Kommandozeile für die neue Applikation

Beispiel: Die laufende Applikation TEST.APP soll nach Beendigung die GEM-Applikation NEU.APP aufrufen, wobei ihr die Zeichenkette „Es war TEST.APP" übergeben wird. Dabei soll die alte Applikation aus dem Speicher entfernt werden.

```
WORD doex, isgr, isover;
BYTE pcmd [128];
BYTE ptail[128];

doex   = TRUE;
isgr   = TRUE;
isover = TRUE;
strcpy (pcmd, "C:\\GEMAPPS\\NEU.APP");
strcpy (ptail, "Es war TEST.APP");
shel_write (doex, isgr, isover, pcmd, ptail);
```

– SHEL_FIND

Hiermit kann eine beliebige Datei gesucht werden. Wird sie gefunden, so wird der komplette Pfad der Datei zurückgeliefert. Die Funktion shel_find sucht im aktuellen Pfad und in allen Pfaden, die in der Environment-Variablen PATH angegeben sind.

Beispiel: Suche nach Dateinamen CDS.DOC. Der Pfad ist wie folgt eingestellt:

```
PATH=C:\;C:\DOS;C;\GEMAPPS;C:\GEMAPPS\WORDPLUS\DOCS

BYTE ppath [128];

strcpy (ppath, "CDS.DOC");
shel_find (ppath);
```

Die Varibale ppath enthält nach dem Aufruf von shel_find den Wert „C:\GEMAPPS\WORDPLUS\DOCS\CDS.DOC".

– SHEL_ENVRN

Damit läßt sich die sogenannte Umgebung (environment) des Betriebssystems auf einen Parameter hin untersuchen. In obigem Beispiel wurde z.B. der Pfad in die Betriebssystem-Umgebung gesetzt. Möchte man z.B. den Wert des Parameters PATH=

wissen, muß shel_envrn aufgerufen werden. Man übergibt den Suchbegriff und erhält einen Zeiger auf das Byte, das dem Suchbegriff folgt.

Beispiel: Suche nach dem Wert des Umgebungsparameters PATH=.

```
BYTE *ppath;
BYTE psrch [128];

strcpy (psrch, "PATH=");
shel_envrn (&ppath, psrch);
```

Die Variable ppath enthält die Zeichenkette „C:\;C:\DOS;C;\GEMAPPS;C:\GEMAPPS\WORDPLUS".

– SHEL_RDEF

Der Dateiname sowie das Inhaltsverzeichnis der Default-Applikation (s.o.) wird ab GEM Version 2.X zurückgegeben. Im Normalfall ist dies DESKTOP.APP.

Beispiel:

```
BYTE lpcmd [32];
BYTE lpdir [128];

shel_rdef (lpcmd, lpdir);
```

– SHEL_WDEF

Durch Aufruf der Funktion shel_wdef kann die Applikation bestimmt werden, die nach Beendigung einer beliebigen Applikation aufgerufen werden soll. Im Normalfall ist die Default-Applikation der GEM-Desktop mit Namen DESKTOP.APP.

Beispiel: Die Applikation C:\GEMAPPS\SHELL.APP soll als neuer Desktop dienen, d.h. man soll von SHELL aus jede andere Applikation aufrufen können und nach Beendigung der aufgerufenen Applikation soll wieder SHELL.APP aktiv werden an Stelle des GEM-Desktop.

```
BYTE lpcmd [32];
BYTE lpdir [128];

strcpy (lpcmd, "SHELL.APP");
strcpy (lpdir, "C:\\GEMAPPS\\");
shel_wdef (lpcmd, lpdir);
```

Achtung: Wird die eigene Shell wieder gestartet, so bekommt sie als Parameter genau die Werte geliefert, die sie selbst an das aufgerufene Programm übergeben hat. Auf diese Parameter sollte die Shell dann nicht oder entsprechend reagieren.

## 12. EXTENDED-GRAPHICS-LIBRARY

| | |
|---|---|
| 130 – xgrf_stepcalc | Rahmenparameter berechnen |
| 131 – xgrf_2box | Sich bewegende Boxen zeichnen |

Da ab GEM 2.X die Funktionen graf_growbox und graf_shrinkbox aus dem AES eliminiert wurden (wahrscheinlich auch wegen des Rechtsstreits mit Apple), bietet das AES diese beiden Ersatzfunktionen an. Bisher wurden sie von keiner Applikation benutzt. Wir zeigen aber, wie mit ihnen die beiden fehlenden Funktionen aus der Graphics-Library nachgebildet werden können.

- XGRF_STEPCALC
- XGRF_2BOX

Mit der Funktion xgrf_stepcalc berechnet man die Schrittweiten in x- und y-Richtung und erhält die Werte, die man für die Funktion xgrf2box benötigt.

Beispiel: Simulation der alten GEM-1.X-Funktionen graf_growbox und graf_shrinkbox.

```
LOCAL VOID graf_growbox (orgx, orgy, orgw, orgh, x, y, w, h)
WORD orgx, orgy, orgw, orgh;
WORD x, y, w, h;

{  WORD  cx, cy, cnt, xstep, ystep;

  xgrf_stepcalc (orgw, orgh, x, y, w, h,
                 &cx, &cy, &cnt, &xstep, &ystep);
  graf_mbox (orgw, orgh, orgx, orgy, cx, cy);
  xgrf_2box (cx, cy, orgw, orgh, TRUE, cnt, xstep, ystep, TRUE);
} /* graf_growbox */

/*****************************************************************/

LOCAL VOID graf_shrinkbox (orgx, orgy, orgw, orgh, x, y, w, h)
WORD orgx, orgy, orgw, orgh;WORD x, y, w, h;

{  WORD cx, cy, cnt, xstep, ystep;

  xgrf_stepcalc (orgw, orgh, x, y, w, h,
                 &cx, &cy, &cnt, &xstep, &ystep);
```

```
  xgrf_2box (x, y, w, h, TRUE, cnt, -xstep, -ystep, TRUE);
  graf_mbox (orgw, orgh, cx, cy, orgx, orgy);
} /* graf_shrinkbox */
#endif

/*************************************************************/
```

Zusammenfassung: Wir haben in diesem Kapitel nach dem VDI auch das AES kennengelernt. Mit beiden Paketen von Funktionen ist es möglich geworden, geräteunabhängige Ausgaben zu produzieren und fensterorientierte Software mit Menüleisten und Dialogboxen zu gestalten und zu implementieren.

Das AES bestand dabei wie das VDI aus verschiedenen Gruppen von Funktionen. Diese Gruppen deckten die Ereignisverarbeitung (Event-Library), das Handling von Fenstern (Window-Library), das Anzeigen von Menübäumen (Menu-Library), den Umgang mit Objekten (Objekt-Library) usw. ab.

Ein wesentlicher Gesichtspunkt war das Trennen der Aufgaben des AES und der Applikation. Während das AES die Verwaltung der Menüleiste und der Randkomponenten der Fenster übernahm, war es Aufgabe der Applikation, das Innere von Fenstern zu zeichnen. Dabei spielte der Begriff der Nachricht (Messages) eine zentrale Rolle. Mit ihnen sendet das AES unserer Applikation Nachrichten (Fenster neu zeichnen, Menü angewählt usw.) und wir können Nachrichten an andere Applikationen (oder Accessories) senden (z.B. an einen Druckerspooler).

Die Form-Library schließlich nahm uns die gesamte Arbeit ab, die Interaktion des Benutzers mit einer Dialogbox zu überwachen. Wir konnten somit nach Beendigung eines Dialogs alle Informationen aus dem Objektbaum der Dialogbox herausholen. Objektbäume haben wir außerdem durch benutzerdefinierte Funktionen erweitern können (Checkboxen, runde Radiobuttons).

In Kapitel 4 wird das kleine GEM-Programm SHOWGEM vorgestellt, das den Umgang mit Dialogboxen und benutzerdefinierten Objekten verdeutlicht. Damit es nicht zu kompliziert wird, wurde auf die Benutzung von Fenstern verzichtet. Es zeigt Metadateien und Bit-Image-Dateien auf beliebigen Ausgabegeräten an. Wer alles genau wissen möchte, muß sich dann in die Beispielapplikation SCRAP einlesen, die alles aus GEM herausholt, was möglich ist. Dabei werden keinerlei Tricks verwendet, sondern nur saubere VDI- und AES-Aufrufe. Wer seine Applikationen so schreibt, wie zum Beispiel SCRAP, kann sicher sein, daß er nicht eine einzige Zeile umprogrammieren muß, wenn er auf ein anderes Betriebssystem oder eine andere GEM-Version umsteigt.

## 2.5 Resourcen

In diesem Kapitel soll noch kurz auf Resource-Dateien eingegangen werden. Es werden ein Programm und ein Modul vorgestellt, die folgende Aufgaben haben:

a) Ein GEM-Resource-Konverter, der Resource-Dateien von MC68000-Prozessoren nach INTEL-Prozessoren überführt und umgekehrt.

b) Das Modul RCM (Resource-Create-Modul), das es erlaubt, die vom RCS erzeugten Dateien mit Suffix RSH (Resource-Header), welche C-Quelltexte von Resourcen darstellen, in ein Programm mit einzubinden.

### 2.5.1 GRC

Dieses Programm kann auf einem ATARI ST übersetzt und angewendet werden. Es wandelt Resource-Dateien von MC68xxx-Prozessoren in Resourcen für Prozessoren vom Typ INTEL 80xxx um. Dabei wird auch der Zeichensatz von ATARI und MS-DOS berücksichtigt. ATARI-Besitzer kennen das Problem ja schon bei Druckern. Das „ß"- und das „§"-Zeichen stimmen nicht mit dem IBM-Zeichensatz überein.

```
/*****************************************************************/
/*                                                               */
/* GEM resource converter for ATARI ST                           */
/*                                                               */
/* used to convert GEM resource files                            */
/* from MC68000 to INTEL format and vice versa                   */
/*                                                               */
/* Version: [1.2]                                                */
/* Date   : 1-February-89                                        */
/* Authors: Jürgen & Dieter Geiß                                 */
/*                                                               */
/*****************************************************************/


#include <stdio.h>
#include <string.h>
#include <portab.h>
#include <vdi.h>
#include <aes.h>

#if GEMDOS
#include <osbind.h>
#else
#include <gemdos.h>
#include <dosbind.h>
#endif
```

```
/****** DEFINES ********************************************************/

#define STRLEN  82
#define MAX_RS  65536L    /* max 64k resource files, must be LONG */

/****** TYPES **********************************************************/

typedef BYTE STRING [STRLEN];

/****** VARIABLES ******************************************************/

WORD   handle;          /* file handle                                  */
LONG   size;            /* size of file                                 */
STRING srcname;         /* source resource filename                     */
STRING destname;        /* destination resource filename                */
STRING direction;       /* 0 = MC68K -> INTEL, 1 = INTEL -> MC68K */
RSHDR  *header;         /* resource header                              */
UBYTE  *rsc_buffer;     /* buffer for resource file                     */

/****** PROTOTYPES *****************************************************/

LOCAL VOID strupper     _((BYTE *s));
LOCAL VOID flip_word    _((BYTE *adr));
LOCAL VOID flip_long    _((WORD *adr));
LOCAL VOID MC68KtoINTEL _((VOID));
LOCAL VOID INTELtoMC68K _((VOID));

/****** FUNCTIONS ******************************************************/

LOCAL VOID strupper (s)
REG BYTE *s;

{
  while (*s)
  {
    if (('a' <= *s) && (*s <= 'z')) *s -= 'a' - 'A';
    s++;
  } /* while */
} /* strupper */

/*********************************************************************/

LOCAL VOID flip_word (adr)
REG BYTE *adr;
```

```
{
  REG BYTE c;

  c       = adr [0];
  adr [0] = adr [1];
  adr [1] = c;
} /* flip_word */

/*****************************************************************/

LOCAL VOID flip_long (adr)
REG WORD *adr;

{
  REG WORD i;

  i       = adr [0];
  adr [0] = adr [1];
  adr [1] = i;

  flip_word ((BYTE *)&adr [0]);
  flip_word ((BYTE *)&adr [1]);
} /* flip_long */

/*****************************************************************/

LOCAL VOID MC68KtoINTEL ()

{
  WORD    i, j, words;
  WORD    *pmask;
  WORD    *pdata;
  OBJECT  *pobject;
  TEDINFO *ptedinfo;
  ICONBLK *piconblk;
  BITBLK  *pbitblk;
  BYTE    **pfrstr;
  BITBLK  *pfrimg;
  OBJECT  **ptrindex;
  UBYTE   *pstring;

  handle = Fopen (srcname, 0);

  if (handle < 0)
  {
    printf ("Error in opening %s\n", srcname);
    return;
  } /* if */
```

```
size = Fread (handle, MAX_RS, rsc_buffer);
Fclose (handle);

/***** flip data and mask of iconblocks *****/

piconblk = (ICONBLK *)((ULONG)rsc_buffer + header->rsh_iconblk);

for (i = 0; i < header->rsh_nib; i++)
{
  words = piconblk [i].ib_wicon / 16 * piconblk [i].ib_hicon;

  pmask = (WORD *)((ULONG)rsc_buffer +
                   (ULONG)piconblk [i].ib_pmask);
  for (j = 0; j < words; j++) flip_word ((BYTE *)&pmask [j]);

  pdata = (WORD *)((ULONG)rsc_buffer +
                   (ULONG)piconblk [i].ib_pdata);
  for (j = 0; j < words; j++) flip_word ((BYTE *)&pdata [j]);
} /* for */

/***** flip data of bitblocks *****/

pbitblk = (BITBLK *)((ULONG)rsc_buffer + header->rsh_bitblk);

for (i = 0; i < header->rsh_nbb; i++)
{
  words = pbitblk [i].bi_wb / 2 * pbitblk [i].bi_hl;

  pdata = (WORD *)((ULONG)rsc_buffer +
                   (ULONG)pbitblk [i].bi_pdata);
  for (j = 0; j < words; j++) flip_word ((BYTE *)&pdata [j]);
} /* for */

/***** flip objects *****/

pobject = (OBJECT *)((ULONG)rsc_buffer + header->rsh_object);

for (i = 0; i < header->rsh_nobs; i++)
{
  flip_word ((BYTE *)&pobject [i].ob_next);
  flip_word ((BYTE *)&pobject [i].ob_head);
  flip_word ((BYTE *)&pobject [i].ob_tail);
  flip_word ((BYTE *)&pobject [i].ob_type);
  flip_word ((BYTE *)&pobject [i].ob_flags);
  flip_word ((BYTE *)&pobject [i].ob_state);
  flip_long ((WORD *)&pobject [i].ob_spec);
```

```
    flip_word ((BYTE *)&pobject [i].ob_x);
    flip_word ((BYTE *)&pobject [i].ob_y);
    flip_word ((BYTE *)&pobject [i].ob_width);
    flip_word ((BYTE *)&pobject [i].ob_height);
  } /* for */

  /***** flip tedinfos *****/

  ptedinfo = (TEDINFO *)((ULONG)rsc_buffer + header->rsh_tedinfo);

  for (i = 0; i < header->rsh_nted; i++)
  {
    flip_long ((WORD *)&ptedinfo [i].te_ptext);
    flip_long ((WORD *)&ptedinfo [i].te_ptmplt);
    flip_long ((WORD *)&ptedinfo [i].te_pvalid);
    flip_word ((BYTE *)&ptedinfo [i].te_font);
    flip_word ((BYTE *)&ptedinfo [i].te_junk1);
    flip_word ((BYTE *)&ptedinfo [i].te_just);
    flip_word ((BYTE *)&ptedinfo [i].te_color);
    flip_word ((BYTE *)&ptedinfo [i].te_junk2);
    flip_word ((BYTE *)&ptedinfo [i].te_thickness);
    flip_word ((BYTE *)&ptedinfo [i].te_txtlen);
    flip_word ((BYTE *)&ptedinfo [i].te_tmplen);
  } /* for */

  /***** flip iconblocks *****/

  piconblk = (ICONBLK *)((ULONG)rsc_buffer + header->rsh_iconblk);

  for (i = 0; i < header->rsh_nib; i++)
  {
    flip_long ((WORD *)&piconblk [i].ib_pmask);
    flip_long ((WORD *)&piconblk [i].ib_pdata);
    flip_long ((WORD *)&piconblk [i].ib_ptext);
    flip_word ((BYTE *)&piconblk [i].ib_char);
    flip_word ((BYTE *)&piconblk [i].ib_xchar);
    flip_word ((BYTE *)&piconblk [i].ib_ychar);
    flip_word ((BYTE *)&piconblk [i].ib_xicon);
    flip_word ((BYTE *)&piconblk [i].ib_yicon);
    flip_word ((BYTE *)&piconblk [i].ib_wicon);
    flip_word ((BYTE *)&piconblk [i].ib_hicon);
    flip_word ((BYTE *)&piconblk [i].ib_xtext);
    flip_word ((BYTE *)&piconblk [i].ib_ytext);
    flip_word ((BYTE *)&piconblk [i].ib_wtext);
    flip_word ((BYTE *)&piconblk [i].ib_htext);
  } /* for */
```

```
/***** flip bitblocks *****/

pbitblk = (BITBLK *)((ULONG)rsc_buffer + header->rsh_bitblk);

for (i = 0; i < header->rsh_nbb; i++)
{
  flip_long ((WORD *)&pbitblk [i].bi_pdata);
  flip_word ((BYTE *)&pbitblk [i].bi_wb);
  flip_word ((BYTE *)&pbitblk [i].bi_hl);
  flip_word ((BYTE *)&pbitblk [i].bi_x);
  flip_word ((BYTE *)&pbitblk [i].bi_y);
  flip_word ((BYTE *)&pbitblk [i].bi_color);
} /* for */

/***** flip free strings *****/

pfrstr = (BYTE **)((ULONG)rsc_buffer + header->rsh_frstr);
for (i = 0; i < header->rsh_nstring; i++)
  flip_long ((WORD *)&pfrstr [i]);

/***** flip free images *****/

pfrimg = (BITBLK *)((ULONG)rsc_buffer + header->rsh_frimg);
for (i = 0; i < header->rsh_nimages; i++)
  flip_long ((WORD *)&pfrimg [i]);

/***** flip trees *****/

ptrindex = (OBJECT **)((ULONG)rsc_buffer + header->rsh_trindex);
for (i = 0; i < header->rsh_ntree; i++)
  flip_long ((WORD *)&ptrindex [i]);

/* fix font from ATARI to IBM */

pstring = (UBYTE *)((ULONG)rsc_buffer + header->rsh_string);

while (pstring < (UBYTE *)((ULONG)rsc_buffer + header->rsh_imdata))
{
  if (*pstring == 221) *pstring =  21;
  if (*pstring == 158) *pstring = 225;
  pstring++;
} /* while */
```

```
  /***** flip_header *****/

  flip_word ((BYTE *)&header->rsh_vrsn);
  flip_word ((BYTE *)&header->rsh_object);
  flip_word ((BYTE *)&header->rsh_tedinfo);
  flip_word ((BYTE *)&header->rsh_iconblk);
  flip_word ((BYTE *)&header->rsh_bitblk);
  flip_word ((BYTE *)&header->rsh_frstr);
  flip_word ((BYTE *)&header->rsh_string);
  flip_word ((BYTE *)&header->rsh_imdata);
  flip_word ((BYTE *)&header->rsh_frimg);
  flip_word ((BYTE *)&header->rsh_trindex);
  flip_word ((BYTE *)&header->rsh_nobs);
  flip_word ((BYTE *)&header->rsh_ntree);
  flip_word ((BYTE *)&header->rsh_nted);
  flip_word ((BYTE *)&header->rsh_nib);
  flip_word ((BYTE *)&header->rsh_nbb);
  flip_word ((BYTE *)&header->rsh_nstring);
  flip_word ((BYTE *)&header->rsh_nimages);
  flip_word ((BYTE *)&header->rsh_rssize);

  handle = Fcreate (destname, 0x0);

  if (handle < 0)
  {
    printf ("Error in creating %s\n", destname);
    return;
  } /* if */

  size = Fwrite (handle, size, rsc_buffer);
  Fclose (handle);
} /* MC68KtoINTEL */

/*************************************************************************/

LOCAL VOID INTELtoMC68K ()

{
  WORD    i, j, words;
  WORD    *pmask;
  WORD    *pdata;
  OBJECT  *pobject;
  TEDINFO *ptedinfo;
  ICONBLK *piconblk;
  BITBLK  *pbitblk;
  BYTE    **pfrstr;
  BITBLK  *pfrimg;
  OBJECT  **ptrindex;
  UBYTE   *pstring;  handle = Fopen (srcname, 0);
```

```
if (handle < 0)
{
  printf ("Error in opening %s\n", srcname);
  return;
} /* if */

size = Fread (handle, MAX_RS, rsc_buffer);
Fclose (handle);

/***** flip_header *****/

flip_word ((BYTE *)&header->rsh_vrsn);
flip_word ((BYTE *)&header->rsh_object);
flip_word ((BYTE *)&header->rsh_tedinfo);
flip_word ((BYTE *)&header->rsh_iconblk);
flip_word ((BYTE *)&header->rsh_bitblk);
flip_word ((BYTE *)&header->rsh_frstr);
flip_word ((BYTE *)&header->rsh_string);
flip_word ((BYTE *)&header->rsh_imdata);
flip_word ((BYTE *)&header->rsh_frimg);
flip_word ((BYTE *)&header->rsh_trindex);
flip_word ((BYTE *)&header->rsh_nobs);
flip_word ((BYTE *)&header->rsh_ntree);
flip_word ((BYTE *)&header->rsh_nted);
flip_word ((BYTE *)&header->rsh_nib);
flip_word ((BYTE *)&header->rsh_nbb);
flip_word ((BYTE *)&header->rsh_nstring);
flip_word ((BYTE *)&header->rsh_nimages);
flip_word ((BYTE *)&header->rsh_rssize);

/* fix font from IBM to ATARI */

pstring = (UBYTE *)((ULONG)rsc_buffer + header->rsh_string);

while (pstring < (UBYTE *)((ULONG)rsc_buffer + header->rsh_imdata))
{
  if (*pstring ==  21) *pstring = 221;
  if (*pstring == 225) *pstring = 158;
  pstring++;
} /* while */

/***** fix objects *****/

pobject = (OBJECT *)((ULONG)rsc_buffer + header->rsh_object);
```

```
for (i = 0; i < header->rsh_nobs; i++)
{
  flip_word ((BYTE *)&pobject [i].ob_next);
  flip_word ((BYTE *)&pobject [i].ob_head);
  flip_word ((BYTE *)&pobject [i].ob_tail);
  flip_word ((BYTE *)&pobject [i].ob_type);
  flip_word ((BYTE *)&pobject [i].ob_flags);
  flip_word ((BYTE *)&pobject [i].ob_state);
  flip_long ((WORD *)&pobject [i].ob_spec);
  flip_word ((BYTE *)&pobject [i].ob_x);
  flip_word ((BYTE *)&pobject [i].ob_y);
  flip_word ((BYTE *)&pobject [i].ob_width);
  flip_word ((BYTE *)&pobject [i].ob_height);
} /* for */

/***** flip tedinfos *****/

ptedinfo = (TEDINFO *)((ULONG)rsc_buffer + header->rsh_tedinfo);

for (i = 0; i < header->rsh_nted; i++)
{
  flip_long ((WORD *)&ptedinfo [i].te_ptext);
  flip_long ((WORD *)&ptedinfo [i].te_ptmplt);
  flip_long ((WORD *)&ptedinfo [i].te_pvalid);
  flip_word ((BYTE *)&ptedinfo [i].te_font);
  flip_word ((BYTE *)&ptedinfo [i].te_junk1);
  flip_word ((BYTE *)&ptedinfo [i].te_just);
  flip_word ((BYTE *)&ptedinfo [i].te_color);
  flip_word ((BYTE *)&ptedinfo [i].te_junk2);
  flip_word ((BYTE *)&ptedinfo [i].te_thickness);
  flip_word ((BYTE *)&ptedinfo [i].te_txtlen);
  flip_word ((BYTE *)&ptedinfo [i].te_tmplen);
} /* for */

/***** flip iconblocks *****/

piconblk = (ICONBLK *)((ULONG)rsc_buffer + header->rsh_iconblk);

for (i = 0; i < header->rsh_nib; i++)
{
  flip_long ((WORD *)&piconblk [i].ib_pmask);
  flip_long ((WORD *)&piconblk [i].ib_pdata);
  flip_long ((WORD *)&piconblk [i].ib_ptext);
  flip_word ((BYTE *)&piconblk [i].ib_char);
  flip_word ((BYTE *)&piconblk [i].ib_xchar);
  flip_word ((BYTE *)&piconblk [i].ib_ychar);
```

```
    flip_word ((BYTE *)&piconblk [i].ib_xicon);
    flip_word ((BYTE *)&piconblk [i].ib_yicon);
    flip_word ((BYTE *)&piconblk [i].ib_wicon);
    flip_word ((BYTE *)&piconblk [i].ib_hicon);
    flip_word ((BYTE *)&piconblk [i].ib_xtext);
    flip_word ((BYTE *)&piconblk [i].ib_ytext);
    flip_word ((BYTE *)&piconblk [i].ib_wtext);
    flip_word ((BYTE *)&piconblk [i].ib_htext);
  } /* for */

  /***** flip bitblocks *****/

  pbitblk = (BITBLK *)((ULONG)rsc_buffer + header->rsh_bitblk);

  for (i = 0; i < header->rsh_nbb; i++)
  {
    flip_long ((WORD *)&pbitblk [i].bi_pdata);
    flip_word ((BYTE *)&pbitblk [i].bi_wb);
    flip_word ((BYTE *)&pbitblk [i].bi_hl);
    flip_word ((BYTE *)&pbitblk [i].bi_x);
    flip_word ((BYTE *)&pbitblk [i].bi_y);
    flip_word ((BYTE *)&pbitblk [i].bi_color);
  } /* for */

  /***** flip free strings *****/

  pfrstr = (BYTE **)((ULONG)rsc_buffer + header->rsh_frstr);
  for (i = 0; i < header->rsh_nstring; i++)
    flip_long ((WORD *)&pfrstr [i]);

  /***** flip free images *****/

  pfrimg = (BITBLK *)((ULONG)rsc_buffer + header->rsh_frimg);
  for (i = 0; i < header->rsh_nimages; i++)
    flip_long ((WORD *)&pfrimg [i]);

  /***** flip trees *****/

  ptrindex = (OBJECT **)((ULONG)rsc_buffer + header->rsh_trindex);
  for (i = 0; i < header->rsh_ntree; i++)
    flip_long ((WORD *)&ptrindex [i]);

  /***** flip data and mask of iconblocks *****/

  piconblk = (ICONBLK *)((ULONG)rsc_buffer + header->rsh_iconblk);
```

```
  for (i = 0; i < header->rsh_nib; i++)
  {
    words = piconblk [i].ib_wicon / 16 * piconblk [i].ib_hicon;

    pmask = (WORD *)((ULONG)rsc_buffer +
                     (ULONG)piconblk [i].ib_pmask);
    for (j = 0; j < words; j++) flip_word ((BYTE *)&pmask [j]);

    pdata = (WORD *)((ULONG)rsc_buffer +
                     (ULONG)piconblk [i].ib_pdata);
    for (j = 0; j < words; j++) flip_word ((BYTE *)&pdata [j]);
  } /* for */

  /***** flip data of bitblocks *****/

  pbitblk = (BITBLK *)((ULONG)rsc_buffer + header->rsh_bitblk);

  for (i = 0; i < header->rsh_nbb; i++)
  {
    words = pbitblk [i].bi_wb / 2 * pbitblk [i].bi_hl;

    pdata = (WORD *)((ULONG)rsc_buffer +
                     (ULONG)pbitblk [i].bi_pdata);
    for (j = 0; j < words; j++) flip_word ((BYTE *)&pdata [j]);
  } /* for */

  handle = Fcreate (destname, 0x0);

  if (handle < 0)
  {
    printf ("Error in creating %s\n", destname);
    return;
  } /* if */

  size = Fwrite (handle, size, rsc_buffer);
  Fclose (handle);
} /* INTELtoMC68K */

/*****************************************************************/

GLOBAL WORD main (argc, argv)
WORD argc;
BYTE *argv [];
```

```
{
  printf ("\n\n***** GEM resource converter [1.2] *****\n\n");

  rsc_buffer = (UBYTE *)Malloc (MAX_RS);
  header     = (RSHDR *)rsc_buffer;

  if (rsc_buffer == NULL)
  {
    printf ("Error in allocating buffer for resource file\n");
    return (1);
  } /* if */

  if (argc < 3)
  {
    printf ("Source resource file (without suffix .RSC): ");
    gets (srcname);
    printf ("Direction: 0 = MC68000 -> INTEL\n");
    printf ("           1 = INTEL   -> MC68000\n");
    printf ("Direction: ");
    gets (direction);
  } /* if */
  else
  {
    strcpy (srcname,   argv [1]);
    strcpy (direction, argv [2]);
  } /* else */

  strcat (srcname, ".RSC");
  strupper (srcname);
  strcpy (destname, srcname);

  if (direction [0] == '0')
    MC68KtoINTEL ();
  else
    INTELtoMC68K ();

  printf ("Conversion finished\n");
  return (0);
} /* main */

/*****************************************************************/
```

Zur Bedienung ist folgendes zu sagen: Startet man das Programm als TOS-Programm ohne Parameter, so kommt zuerst die Abfrage nach der Resource-Datei. Dort muß der Name ohne das Suffix „.RSC“ angegeben werden.

Anschließend erfolgt die Abfrage, in welche Richtung die Datei konvertiert werden soll:

Eine Angabe einer 0 konvertiert die Datei vom 68000-Format ins INTEL-Format, eine 1 vom INTEL-Format ins 68000-Format.

Das Programm kann auch von einer Kommando-Shell aus gestartet werden. Man übergibt den Namen und die Transformationsrichtung. Beispiel: Wandeln der Datei SHOWGEM.RSC vom 68000- ins INTEL-Format.

```
grc showgem 0
```

Wird die Konvertierung erfolgreich abgeschlossen, erscheint die Meldung: „Conversion finished".

Die angegebene Datei wird unter dem gleichen Namen konvertiert, so daß nach obigem Beispiel die Datei SHOWGEM.RSC jetzt im INTEL-Format vorliegt. Hat man versehentlich seine einzige Resource-Datei auf diese Weise bearbeitet, so genügt eine Rücktransformation in die andere Richtung. Mathematisch ausgedrückt heißt dies, daß die Rücktransformation einer Transformation einer Resource-Datei wieder die Ursprungsdatei ergibt. Jede Transformation ist also das inverse Element der Rücktransformation.

Zum Algorithmus ist zu sagen, daß in der Transformation von MC68000 nach INTEL der Kopf der Resource-Datei zuletzt umgewandelt wird, da dieser bis zum Schluß für die Berechnungen der Zeiger auf die Objektstrukturen benötigt wird. In der anderen Richtung (INTEL nach MC68000) wird zunächst der Kopf umgewandelt, damit man ihn in der 68000-Prozessorumgebung benutzen kann. Das Programm GRC ist ein anschauliches Beispiel für die Benutzung des Kopfes einer Resource-Datei. Man sieht sehr schön, wie man sich die Informationen zu den einzelnen Objekttypen beschaffen kann.

### 2.5.2 RCM

Um zu verstehen, wie das Resource-Create-Modul arbeitet, muß man wissen, wie die AES-Funktion rsrc_load auf eine Resource-Datei nach dem Laden wirkt.

Nachdem rsrc_load den Kopf der Resource-Datei geladen hat, um deren Länge (rsh_rssize) festzustellen, alloziert es dynamisch Speicher und lädt den Rest der Datei ein. Der Kopf einer Resource-Datei ist in C wie folgt definiert (siehe AES.H):

```
typedef struct rshdr
{
  UWORD rsh_vrsn;
  UWORD rsh_object;
  UWORD rsh_tedinfo;
  UWORD rsh_iconblk;        /* list of ICONBLKS          */
  UWORD rsh_bitblk;
```

```
  UWORD rsh_frstr;
  UWORD rsh_string;
  UWORD rsh_imdata;        /* image data                 */
  UWORD rsh_frimg;
  UWORD rsh_trindex;
  UWORD rsh_nobs;          /* counts of various structs */
  UWORD rsh_ntree;
  UWORD rsh_nted;
  UWORD rsh_nib;
  UWORD rsh_nbb;
  UWORD rsh_nstring;
  UWORD rsh_nimages;
  UWORD rsh_rssize;        /* total bytes in resource   */
} RSHDR;
```

Danach werden folgende Anpassungen vorgenommen:

a) Änderungen der Werte ob_x, ob_y, ob_width und ob_height an die momentane Bildschirmauflösung

b) Modifizieren der Zeiger aus den Strukturen OBJECT (ob_spec), TEDINFO (te_ptext, te_ptmplt, te_pvalid), BITBLK (bi_pdata) und ICONBLK (ib_pmask, ib_pdata, ib_ptext).

c) Aufbauen eines Feldes von Zeigern, die auf jeden Objektbaum verweisen.

d) Speichern der Adresse des Feldes aus c) im AES-Feld global [5] und global [6].

Genau diese Schritte (außer d), führt das Modul RCM ebenfalls aus. Dazu benutzt es u.a. die Funktion rsrc_obfix, die die Anpassung aus a) automatisch vornimmt (siehe auch Modul RESOURCE aus der Applikation SCRAP).

Die Header-Datei RCM.H, die man inkludieren muß, sieht wie folgt aus:

```
/*****************************************************************/
/*                                                               */
/* Modul: RCM.H                                                  */
/* Datum: 16/04/89                                               */
/*                                                               */
/*****************************************************************/

#ifndef __RCM__
#define __RCM__

/****** DEFINES ***************************************************/
```

```
/****** TYPES *********************************************************/

typedef struct
{
  WORD dummy;
  WORD *image;
} RS_IMDOPE;

/****** VARIABLES *****************************************************/

/****** FUNCTIONS *****************************************************/

#if ANSI
GLOBAL VOID rsc_create (WORD      tree,
                        WORD      n_obs,
                        WORD      n_frstr,
                        WORD      n_frimg,
                        BYTE      **rs_strings,
                        LONG      *rs_frstr,
                        BITBLK    *rs_bitblk,
                        LONG      *rs_frimg,
                        ICONBLK   *rs_iconblk,
                        TEDINFO   *rs_tedinfo,
                        OBJECT    *rs_object,
                        OBJECT    **rs_trindex,
                        RS_IMDOPE *rs_imdope);
#else

GLOBAL VOID rsc_create ();
#endif

#endif /* __RCM__ */
```

Die Implemetierung des Moduls in C lautet:

```
/*********************************************************************/
/*                                                                   */
/* Modul: RCM.C                                                      */
/* Datum: 18/05/89                                                   */
/*                                                                   */
/*********************************************************************/

#include <portab.h>
#include <aes.h>
```

```
#include "rcm.h"

/****** DEFINES ********************************************************/

/****** TYPES **********************************************************/

/****** VARIABLES ******************************************************/

/****** FUNCTIONS ******************************************************/

/****** PROTOTYPES *****************************************************/

LOCAL VOID fix_tree    _((WORD n_tree, OBJECT **rs_trindex,
                          OBJECT *rs_object));
LOCAL VOID fix_tedinfo _((WORD object, OBJECT *rs_object,
                          TEDINFO *rs_tedinfo, BYTE **rs_strings));
LOCAL VOID fix_bitblk  _((WORD object, OBJECT *rs_object,
                          BITBLK *rs_bitblk, RS_IMDOPE *rs_imdope));
LOCAL VOID fix_string  _((WORD object, OBJECT *rs_object,
                          BYTE **rs_strings));
LOCAL VOID fix_iconblk _((WORD object, OBJECT *rs_object,
                         ICONBLK *rs_iconblk, RS_IMDOPE *rs_imdope,
                         BYTE **rs_strings));
LOCAL VOID fix_frimg   _((WORD object, LONG *rs_frimg,
                          BITBLK *rs_bitblk, RS_IMDOPE *rs_imdope));

/*********************************************************************/

LOCAL VOID fix_tree (n_tree, rs_trindex, rs_object)
REG WORD   n_tree;
REG OBJECT **rs_trindex;
REG OBJECT *rs_object;

{
  REG WORD   tree;     /* index for trees */
  REG WORD   object;   /* index for objects */
  REG WORD   help;
  REG OBJECT *pobject;

  for (tree = 0; tree < n_tree; tree++) /* fix trees */
  {
    help = (WORD)rs_trindex [tree];
    rs_trindex [tree] = (OBJECT *)&rs_object [help];
```

```
    object  = 0;
    pobject = rs_trindex [tree];

    while (! (pobject [object].ob_flags & LASTOB))
    {
      rsrc_obfix (pobject, object); /* fix xywh of objects */
      object++;
    } /* while */
    rsrc_obfix (pobject, object); /* at least one object */
  } /* for */
} /* fix_tree */

/*****************************************************************/

LOCAL VOID fix_tedinfo (object, rs_object, rs_tedinfo, rs_strings)
REG WORD    object;
REG OBJECT  *rs_object;
REG TEDINFO *rs_tedinfo;
REG BYTE    **rs_strings;

{
  REG WORD index;

  index = (WORD)rs_object [object].ob_spec;
  rs_object [object].ob_spec = (LONG)&rs_tedinfo [index];

  rs_tedinfo [index].te_ptext  =
    rs_strings [(WORD)rs_tedinfo [index].te_ptext];

  rs_tedinfo [index].te_ptmplt =
    rs_strings [(WORD)rs_tedinfo [index].te_ptmplt];

  rs_tedinfo [index].te_pvalid =
    rs_strings [(WORD)rs_tedinfo [index].te_pvalid];
} /* fix_tedinfo */

/*****************************************************************/

LOCAL VOID fix_bitblk (object, rs_object, rs_bitblk, rs_imdope)
REG WORD      object;
REG OBJECT    *rs_object;
REG BITBLK    *rs_bitblk;
REG RS_IMDOPE *rs_imdope;
```

```
{
  REG WORD index1;
  REG WORD index2;

  index1 = (WORD)rs_object [object].ob_spec;
  index2 = (WORD)rs_bitblk [index1].bi_pdata;

  rs_bitblk [index1].bi_pdata = rs_imdope [index2].image;
  rs_object [object].ob_spec  = (LONG)&rs_bitblk [index1];
} /* fix_bitblk */

/*****************************************************************/

LOCAL VOID fix_string (object, rs_object, rs_strings)
REG WORD    object;
REG OBJECT  *rs_object;
REG BYTE    **rs_strings;

{
  rs_object [object].ob_spec = (LONG)rs_strings
                               [(WORD)rs_object [object].ob_spec];
} /* fix_string */

/*****************************************************************/

LOCAL VOID fix_iconblk (object, rs_object, rs_iconblk,
                        rs_imdope, rs_strings)
REG WORD      object;
REG OBJECT    *rs_object;
REG ICONBLK   *rs_iconblk;
REG RS_IMDOPE *rs_imdope;
REG BYTE      **rs_strings;

{
  REG WORD index1;
  REG WORD index2;

  index1 = (WORD)rs_object [object].ob_spec;

  index2 = (WORD)rs_iconblk [index1].ib_pmask;
  rs_iconblk [index1].ib_pmask = rs_imdope [index2].image;

  index2 = (WORD)rs_iconblk [index1].ib_pdata;
  rs_iconblk [index1].ib_pdata = rs_imdope [index2].image;
```

```
  index2 = (WORD)rs_iconblk [index1].ib_ptext;
  rs_iconblk [index1].ib_ptext = rs_strings [index2];

  rs_object [object].ob_spec = (LONG)&rs_iconblk [index1];
} /* fix_iconblk */

/*****************************************************************/

LOCAL VOID fix_frimg (object, rs_frimg, rs_bitblk, rs_imdope)REG WORD
      object;REG LONG
      *rs_frimg;REG BITBLK
      *rs_bitblk;REG RS_IMDOPE *rs_imdope;

{
  REG WORD index1;
  REG WORD index2;

  index1 = (WORD)rs_frimg [object];
  index2 = (WORD)rs_bitblk [index1].bi_pdata;

  rs_bitblk [index1].bi_pdata = rs_imdope [index2].image;
  rs_frimg [object]           = (LONG)&rs_bitblk [index1];
} /* fix_frimg */

/*****************************************************************/

GLOBAL VOID rsc_create (n_tree, n_obs, n_frstr, n_frimg,
                        rs_strings, rs_frstr, rs_bitblk,
                        rs_frimg, rs_iconblk, rs_tedinfo,
                        rs_object, rs_trindex, rs_imdope)
WORD      n_tree;
WORD      n_obs;
WORD      n_frstr;
WORD      n_frimg;
BYTE      **rs_strings;
LONG      *rs_frstr;
BITBLK    *rs_bitblk;
LONG      *rs_frimg;
ICONBLK   *rs_iconblk;
TEDINFO   *rs_tedinfo;
OBJECT    *rs_object;
OBJECT    **rs_trindex;
RS_IMDOPE *rs_imdope;

{  REG WORD object; /* index for objects */
```

```
  fix_tree (n_tree, rs_trindex, rs_object);

  for (object = 0; object < n_obs; object ++) /* fix objects */
  {
    switch (rs_object [object].ob_type & 0xFF)
    {
      case G_BOX      : break;
      case G_TEXT     : fix_tedinfo (object, rs_object, rs_tedinfo,
                                     rs_strings);
                        break;
      case G_BOXTEXT  : fix_tedinfo (object, rs_object, rs_tedinfo,
                                     rs_strings);
                        break;
      case G_IMAGE    : fix_bitblk (object, rs_object, rs_bitblk,
                                    rs_imdope);
                        break;
      case G_PROGDEF  : break;
      case G_IBOX     : break;
      case G_BUTTON   : fix_string (object, rs_object, rs_strings);
                        break;
      case G_BOXCHAR  : break;
      case G_STRING   : fix_string (object, rs_object, rs_strings);
                        break;
      case G_FTEXT    : fix_tedinfo (object, rs_object, rs_tedinfo,
                                     rs_strings);
                        break;
      case G_FBOXTEXT : fix_tedinfo (object, rs_object, rs_tedinfo,
                                     rs_strings);
                        break;
      case G_ICON     : fix_iconblk (object, rs_object, rs_iconblk,
                                     rs_imdope, rs_strings);
                        break;
      case G_TITLE    : fix_string (object, rs_object, rs_strings);
                        break;
    } /* switch */
  } /* for */

  for (object = 0; object < n_frstr; object++)
    rs_frstr [object] = (LONG)rs_strings [rs_frstr [object]];

  for (object = 0; object < n_frimg; object++)
    fix_frimg (object, rs_frimg, rs_bitblk, rs_imdope);
} /* rsc_create */

/*****************************************************************/
```

Statt rsrc_load kann dann rsc_create aufgerufen werden, wobei die Resourcen im Programmcode fest verankert sind. Dies ist bei Verwendung von Accessories ratsam.

## 2.6 Accessories

Accessories sind GEM-Programme, die beim Starten des GEM geladen werden und dann permanent im Speicher bleiben. Sie laufen im Hintergrund einer Applikation ab. Beispiele für Accessories sind: Druckerspooler, Rechner, Uhr, Kontrollfeld usw. Im Gegensatz zu einer Applikation kann man Accessories nie beenden. Sie müssen also in einer Endlosschleife auf Nachrichten vom Bildschirmmanager warten und diese dann ausführen.

Beispiel:

```
while (TRUE)
{
  event = evnt_mesag (msg_buf);
  ...
} /* while */
```

Der einzige Unterschied zu „normalen" GEM-Applikationen besteht darin, daß sie direkt über den Menüpunkt „Desk" links oben (GEM 1.X) bzw. über den Menüpunkt des Namens der Hauptapplikation (ab GEM 2.X), der sich rechts oben in der Menüzeile befindet, aufgerufen werden können.

Accessories können unter GEM 1.X allerdings keine Menüleiste haben. Dies ist ein gravierender Nachteil. Aus diesem Grund wurde im Windowmanager der Beispielapplikation SCRAP ermöglicht, Menüzeilen innerhalb eines GEM-Fensters zu verwalten. Diese Menüzeilen werden ebenfalls mit dem RCS erstellt.

Ab GEM 2.X können Accessories den AES-Aufruf menu_bar tätigen und eine eigene Menüleiste installieren. Jedesmal, wenn ein Fenster des Accessories nach oben kommt, ändert der Bildschirmmanager die Menüzeile auf die des Accessories. Kommt ein Fenster der laufenden Applikation hoch, schaltet er wieder auf die Menüzeile der Applikation um.

## 2.7 Formate

In diesem Kapitel werden die 3 wichtigen GEM-Formate erläutert. Es handelt sich um Metadateien (.GEM), Bit-Image-Dateien (.IMG) und um Ausgabedateien (.OUT). Gerade diese drei Dateien ermöglichen es, Datenaustausch zwischen beliebigen Programmen vorzunehmen.

### 2.7.1 Metadatei-Format

Metadateien sind nichts anderes als abgespeicherte VDI-Befehle, die in objektorientierten Zeichenprogrammen wie GEMDRAW benutzt werden. Im Abschnitt 2.3 wurde genau erläutert, wie das VDI aufgerufen wird: Durch die Bindings werden die Felder contrl, initin, ptsin gefüllt und dann das VDI direkt aufgerufen. Diese drei Felder beschreiben einen VDI-Befehl vollständig.

Im Feld contrl [0] steht die Funktionsnummer, in contrl [1] die Anzahl der Punkte (z.B. bei v_pline), in contrl [3] die Anzahl der Integer-Werte (z.B. bei vst_point) und in contrl [5] steht der sogenannte Subcode, falls es sich um grafische Grundfunktionen (Funktionsnummer 11) oder um ESC-Funktionen (Funktionsnummer 5) handelt.

Je nach Wert der Felder contrl [1] und contrl [3] folgen in einer Metadatei die Koordinatenpunkte im Feld ptsin und/oder die ganzzahligen Werte im Feld intin. Ein Kommando besteht also mindestens aus den contrl-Elementen und, wenn nötig, aus ptsin- und initin-Elementen.

Eine Vorbemerkung sei an dieser Stelle gegeben: Metadateien liegen immer im INTEL-Format (low-high) vor. Möchte man in einem 68000-System Metadateien einlesen, so muß in jedem Wort das nieder- und höherwertige Byte vertauscht werden.

Eine Metadatei beginnt mit einem Kopf, der die Charakteristika der in ihr enthaltenen Grafik beschreibt. Nach dem Kopf kommen 18 VDI-Grundeinstellungen, die beim Öffnen einer Metadatei (v_opnwk) automatisch geschrieben werden. Ab Befehl 19 beginnen die eigenen VDI-Aufrufe. Der Kopf einer Metadatei hat in C folgende Struktur:

```
typedef struct meta_header
{
  WORD id;
  WORD headlen;
  WORD version;
  WORD transform;
  WORD min_x;
  WORD min_y;
  WORD max_x;
  WORD max_y;
  WORD pwidth;
  WORD pheight;
  WORD ll_x;
  WORD ll_y;
  WORD ur_x;
  WORD ur_y;
  WORD bit_image;
} META_HEADER;
```

Wird eine Grafik auf einem beliebigen Computer erstellt, so muß man angeben, wo der Nullpunkt der Grafik zu finden ist. Dieser berechnet sich aus den Werten lower-left-x (ll_x), lower-left-y (ll_y), upper-right-x (ur_x) und upper-right-y (ur_y). Man sagt, daß diese Werte das Koordinatensystem einer Metadatei beschreiben. Die Applikation GEMDRAW legt z.B. den Mittelpunkt einer Grafik auf die Mitte einer Seite, so daß auch negative Koordinatenpunkte (am linken Rand einer Seite) auftreten können.

Um zu bestimmen, wie groß ein Pixel in einer Metdatei sein soll, muß die Seitengröße (pwidth und pheight) angegeben werden. Ist das Koordinatensystem z.B. $1000 \times 1000$ Punkte groß und hat die Seite $10 \times 10$ mm, so ist ein Pixel 1/100 mm groß. Die Seitengröße wird in 1/10 mm angegeben. Eine Seitengröße von $19.05 \times 25.4$ cm ($7.5 \times 10$ Zoll) wird dann durch pwidth = 1905 und pheight = 2540 beschrieben.

Eine Metadatei fängt immer mit $-1$ an, id hat also den Wert 0xFFFF. Das zweite Wort gibt die Länge des Kopfes an (headlen). Addiert man zum Dateianfang die headlen, so kommt man auf den ersten VDI-Befehl.

Der Wert von Version hängt davon ab, mit welchem GEM-System die Metadatei erstellt wurde. Meist ist der Wert 101 (Version 1.1) oder 301 (Version 3.1) ab GEM-Version 3.11. Die Version wird durch die Formel 100 * Hauptversionsnummer + Nebenversionsnummer beschrieben.

Der Wert transform legt fest, ob das NDC oder RC-Koordinatensystem benutzt werden soll. Die Angabe bezieht sich aber nur auf die Orientierung der Y-Werte. Im NDC-System sind positive Y-Werte am oberen Bildrand, im RC-System am unteren Bildrand. Wie groß letztendlich die X- und Y-Werte sein können, hängt nur vom Koordinatensystem ab (ll_x, ll_y, ur_x, ur_y).

Damit ein Grafikprogramm schnell bestimmen kann, ob die Grafik, die in einer Metadatei dargestellt ist, in ein vorgegebenes Rechteck paßt, kann es die Werte min_x, min_y, max_x und max_y benutzen. Es gibt die maximale Ausdehnung der Grafik im benutzten Koordinatensystem an.

Beispiel: Stellen wir uns einen Bildschirm mit $1000 \times 500$ Punkten vor und zeichnen eine Linie vom Punkt 100,200 zu Punkt 300,100. Die Ausdehnung dieser Grafik beträgt dann:

```
min_x = 100;
min_y = 100;
max_x = 300;
max_y = 200;
```

Im Programm VDITEST in Kapitel 2.3 wurden diese Werte für das gezeichnete Haus berechnet. Dort wurde auch gezeigt, wie man eine Metadatei erstellt.

Am Ende des Kopfes befindet sich noch das sogenanne Flag bit_image. Es gibt an, ob sich eine Bit-Imgage-Datei in der Metadatei befindet. In der Realität befindet sich in

keiner Metadatei eine Pixelgrafik. Zeichnet man ein Bild namens TEST in GEMPAINT, so erstellt es außer der Datei TEST.IMG noch die Datei TEST.GEM. In letzterer befindet sich genau ein VDI-Befehl: v_bit_image. Im Kopf ist dann der Wert bit_image auf 1 gesetzt.

Im folgenden wird das Programm DUMPGEM vorgestellt. Es listet zu jeder Metadatei den Kopf und alle darin befindlichen VDI-Befehle im Klartext auf. Es berücksichtigt alle Funktionen bis GEM 3.11 einschließlich Bezier-Funktionen und die internen Funktionen, die von GEMDRAW benutzt werden. Zum Verständnis von Metadateien sollten Sie sich alle auf der Begleitdiskette befindliche Dateien mit Suffix „.GEM" mit diesem Programm ansehen. Achten Sie dann besonders auf die Befehlsfolge ab Befehl Nummer 19.

```
#include <stdio.h>
#include <portab.h>
#include <string.h>

#if GEMDOS
#include <osbind.h>
#else
#include <gemdos.h>
#include <dosbind.h>
#endif

#define MAX_MLEN  65535L /* max metafile length       */
#define MAX_VDI     200  /* vdi functions             */
#define MAX_ESC     200  /* escape functions          */
#define MAX_GDP      20  /* drawing primitives        */
#define MAX_VWM     200  /* v_write_meta subfunctions */
#define MAX_BEZ      20  /* max bezier list           */

#define OP_ESC        5
#define OP_BEZ        6
#define OP_BEZFILL    9
#define OP_GDP       11  /* watch for v_bez_on/off       */
#define BEZ_SUB      13  /* subcode for bezier functions */
#define VWM_SUBCODE  99  /* subcode for v_write_meta     */

#define V_SETRGBI 18500  /* subcode for v_setrgbi */
#define V_TOPBOT  18501  /* subcode for v_topbot  */

/****** Typen ************************************************/

typedef struct metaheader
{
  WORD id;
  WORD headlen;
```

```
  WORD version;
  WORD transform;
  WORD min_x;
  WORD min_y;
  WORD max_x;
  WORD max_y;
  WORD pwidth;
  WORD pheight;
  WORD ll_x;
  WORD ll_y;
  WORD ur_x;
  WORD ur_y;
  WORD bit_image;
} METAHEADER;

/****** Variablen ***************************************************/

LOCAL WORD       contrl [12];
LOCAL WORD       intin [256];
LOCAL WORD       ptsin [256];
LOCAL WORD       n_pts;
LOCAL WORD       n_int;

LOCAL BYTE       namebuffer [32767];
LOCAL BYTE       meta_name [80];
LOCAL ULONG      meta_len;
LOCAL UWORD      meta_index;
LOCAL UWORD      *meta_buffer;
LOCAL METAHEADER *meta_header;

LOCAL BYTE       *vdi_list [MAX_VDI];
LOCAL BYTE       *esc_list [MAX_ESC];
LOCAL BYTE       *gdp_list [MAX_GDP];
LOCAL BYTE       *vwm_list [MAX_VWM];
LOCAL BYTE       *bez_list [MAX_BEZ];
LOCAL BYTE       *undefined  = "undefined";
LOCAL BYTE       *vwm_text   =
         "v_write_meta (vdi_handle, num_ints, ints, num_pts, pts)";
LOCAL BYTE       *ndc_rc [3] = {"NDC", "reserved", "RC"};

/****** Prototypen **************************************************/

LOCAL BOOLEAN file_exist   _((BYTE *filename));
LOCAL VOID    strupper     _((BYTE *s));
LOCAL VOID    flip_word    _((BYTE *adr));
LOCAL BOOLEAN get_list     _((VOID));
```

```
LOCAL BOOLEAN read_meta    _((BYTE *meta_name));
LOCAL VOID    show_header  _((VOID));
LOCAL WORD    get_word     _((VOID));
LOCAL BOOLEAN get_code     _((VOID));
LOCAL BYTE    *get_command _((VOID));
LOCAL VOID    show_meta    _((VOID));

/*****************************************************************/

LOCAL BOOLEAN file_exist (filename)
BYTE *filename;

{
#if GEMDOS
  return (Fsfirst (filename, 0x00) == 0);
#else
  return (Fsfirst (filename, 0x00) > 0);
#endif
} /* file_exist */

/*****************************************************************/

LOCAL VOID strupper (s)
BYTE *s;

{
  while (*s)
  {
    if (*s >= 'a') *s &= 0xDF;
    s++;
  } /* while */
} /* strupper */

/*****************************************************************/

LOCAL VOID flip_word (adr)
BYTE *adr;

{  BYTE c;

  c      = adr [0];
  adr [0] = adr [1];
  adr [1] = c;
} /* flip_word */

/*****************************************************************/
```

```
LOCAL BOOLEAN get_list ()

{
  FILE    *fp;
  BYTE    *funcname;
  BYTE    s [255];
  WORD    i, intin_0;
  WORD    opcode, subcode;
  BOOLEAN ok;

  fp = fopen ("DUMPGEM.DAT", "r");

  if (fp == NULL)
  {
    printf ("DUMPGEM.DAT not found\n");
    ok = FALSE;
  } /* if */
  else
  {
    for (i = 0; i < MAX_VDI; i++) vdi_list [i] = undefined;
    for (i = 0; i < MAX_ESC; i++) esc_list [i] = undefined;
    for (i = 0; i < MAX_GDP; i++) gdp_list [i] = undefined;
    for (i = 0; i < MAX_VWM; i++) vwm_list [i] = vwm_text;
    for (i = 0; i < MAX_BEZ; i++) bez_list [i] = undefined;

    funcname = namebuffer;

    while (fgets (s, 255, fp) != NULL)
    {
      if (*s != '/')
      {
        sscanf (s, "%d,", &opcode);
        sscanf (s +  4, "%d,", &subcode);
        sscanf (s + 10, "%d,", &intin_0);
        strcpy (funcname, s + 15);
        funcname [strlen (funcname) - 1] = EOS; /* cut off \n */

        if (opcode == OP_ESC ||
            opcode == OP_GDP ||
            opcode == OP_BEZ ||
            opcode == OP_BEZFILL)
        {
          switch (opcode)
          {
```

```
            case OP_ESC     : if (subcode == V_SETRGBI)
                                subcode = 1
                                if (subcode == V_TOPBOT)
                                subcode = 151;

                              if (subcode == VWM_SUBCODE)
                                vwm_list [intin_0] = funcname;
                              else
                                esc_list [subcode] = funcname;
                              break;
            case OP_GDP     : if (subcode == BEZ_SUB)
                              {
                                if (intin_0 == 0) /* v_bez_off */
                                  gdp_list [MAX_GDP - 1] = funcname;
                                else /* v_bez_on */
                                  gdp_list [MAX_GDP - 2] = funcname;
                              } /* if */
                              else
                                gdp_list [subcode] = funcname;
                              break;
            case OP_BEZ     : if (subcode == BEZ_SUB)
                                bez_list [opcode] = funcname;
                              else
                                vdi_list [opcode] = funcname;
                              break;
            case OP_BEZFILL : if (subcode == BEZ_SUB)
                                bez_list [opcode] = funcname;
                              else
                                vdi_list [opcode] = funcname;
                              break;
          } /* switch */
        } /* if */
        else
          vdi_list [opcode] = funcname; /* no special case */
      } /* if */

      funcname += strlen (funcname) + 1;
    } /* while */

    ok = TRUE;
  } /* if */

  return (ok);
} /* get_list */

/*****************************************************************/
```

```
LOCAL BOOLEAN read_meta (meta_name)
BYTE *meta_name;


{
  WORD    handle;
  WORD    i;


  meta_len    = MAX_MLEN;
  meta_buffer = (UWORD *)Malloc (meta_len);


  if (meta_buffer != NULL)
  {
    if (file_exist (meta_name))
    {
      handle   = Fopen (meta_name, 0);
      meta_len = Fread (handle, meta_len, meta_buffer);


#if GEMDOS
      for (i = 0; i < meta_len / 2; i++)
        flip_word ((BYTE *)&meta_buffer [i]);
#endif


      meta_header = (METAHEADER *)meta_buffer;
      meta_index  = meta_header->headlen;


      Fclose (handle);
      return (TRUE);
    } /* if */
    else
    {
      printf ("%s not found\n", meta_name);
      return (FALSE);
    } /* else */
  } /* if */
  else
  {
    printf ("Not enough memory to allocate buffer\n");
    return (FALSE);
  } /* else */
} /* read_meta */


/*****************************************************************/
```

```
LOCAL VOID show_header ()

{  printf ("\nMetafile: %s\n\n", meta_name);

  printf ("metafile id           : %6d\n", meta_header->id);
  printf ("header length         : %6d\n", meta_header->headlen);
  printf ("GEM version           : %4d.%d\n",
          meta_header->version / 100, meta_header->version % 100);
  printf ("NDC/RC flag           : %6s\n",
          ndc_rc [meta_header->transform]);
  printf ("extent min x          : %6d\n", meta_header->min_x);
  printf ("extent min y          : %6d\n", meta_header->min_y);
  printf ("extent max x          : %6d\n", meta_header->max_x);
  printf ("extent max y          : %6d\n", meta_header->max_y);
  printf ("page width            : %3d.%02d cm\n",
          meta_header->pwidth / 100, meta_header->pwidth % 100);
  printf ("page height           : %3d.%02d cm\n",
          meta_header->pheight / 100, meta_header->pheight % 100);
  printf ("window lower left x   : %6d\n", meta_header->ll_x);
  printf ("window lower left y   : %6d\n", meta_header->ll_y);
  printf ("window upper right x  : %6d\n", meta_header->ur_x);
  printf ("window upper right y  : %6d\n", meta_header->ur_y);
  printf ("bit image opcode flag : %6d\n", meta_header->bit_image);
  printf ("\n\n");
} /* show_header */

/*********************************************************************/

LOCAL WORD get_word ()

{
  if (meta_index > meta_len) return (-1);
  return (meta_buffer [meta_index++]);
} /* get_word */

/*********************************************************************/

LOCAL BOOLEAN get_code ()

{
  WORD i;

  contrl [0] = get_word ();
  if (contrl [0] == -1) return (FALSE);
```

```
  contrl [1] = get_word ();
  contrl [3] = get_word ();
  contrl [5] = get_word ();

  n_pts = contrl [1] * 2;
  n_int = contrl [3];

  for (i = 0; i < n_pts; i++) ptsin [i] = get_word ();
  for (i = 0; i < n_int; i++) intin [i] = get_word ();

  return (TRUE);
} /* get_code */

/*****************************************************************/

LOCAL BYTE *get_command ()

{
  BYTE *command;
  WORD opcode;
  WORD subcode;

  opcode  = contrl [0];
  subcode = contrl [5];

  if (opcode == OP_ESC ||
      opcode == OP_GDP ||
      opcode == OP_BEZ ||
      opcode == OP_BEZFILL)
  {
    switch (opcode)
    {
      case OP_ESC     : if (subcode == V_SETRGBI) subcode = 150;
                        if (subcode == V_TOPBOT)  subcode = 151;

                        if (subcode == VWM_SUBCODE)
                          command = vwm_list [intin [0]];
                        else
                          command = esc_list [subcode];
                        break;
      case OP_GDP     : if (subcode == BEZ_SUB)
                        {
                          if (contrl [1] == 0) /* v_bez_off */
                            command = gdp_list [MAX_GDP - 1];
                          else /* v_bez_on */
                            command = gdp_list [MAX_GDP - 2];
```

```
                        n_pts = 0;
                      } /* if */
                      else
                        command =  gdp_list [subcode];
                      break;
      case OP_BEZ     : if (subcode == BEZ_SUB)
                        command = bez_list [opcode];
                      else
                        command = vdi_list [opcode];
                      break;
      case OP_BEZFILL : if (subcode == BEZ_SUB)
                        command = bez_list [opcode];
                      else
                        command = vdi_list [opcode];
                      break;
    } /* switch */
  } /* if */
  else
    command = vdi_list [opcode]; /* no special case */

  return (command);
} /* get_command */

/*************************************************************************/

LOCAL VOID show_meta ()

{
  WORD  count;
  WORD  i;
  BYTE *command;

  count = 0;

  while (get_code ())
  {
    count++;
    command = get_command ();
    printf ("***** Command #%d *****\n\n", count);
    printf ("%s\n", command);

    printf ("contrl [0]: %d\n", contrl [0]);
    printf ("contrl [1]: %d\n", contrl [1]);
    printf ("contrl [3]: %d\n", contrl [3]);
    printf ("contrl [5]: %d\n", contrl [5]);
    printf ("\n");
```

```
    if (n_pts > 0)
    {
      for (i = 0; i < n_pts; i += 2)
      {
        printf ("ptsin [%d] = x = %d\n", i, ptsin [i]);
        printf ("ptsin [%d] = y = %d\n", i + 1, ptsin [i + 1]);
      } /* for */
      printf ("\n");
    } /* if */

    if (n_int > 0)
    {
      for (i = 0; i < n_int; i++)
        printf ("intin [%d] = %d\n", i, intin [i]);
      printf ("\n");
    } /* if */

    printf ("\n");
  } /* while */
} /* show_meta */

/*****************************************************************/

GLOBAL WORD main (argc, argv)
WORD argc;BYTE *argv [];

{
  if (argc < 2)
  {
    printf ("Dump which metafile (without suffix .GEM): ");
    gets (meta_name);
  } /* if */
  else
    strcpy (meta_name, argv [1]);

  if (*meta_name)
  {
    strupper (meta_name);
    strcat (meta_name, ".GEM");

    if (get_list () && read_meta (meta_name))
    {
      show_header ();
      show_meta ();
    } /* if */
  } /* if */
```

```
  return (0);
} /* main */

/*****************************************************************/
```

Beim Start des Programmes DUMPGEM muß sich die Datei DUMPGEM.DAT im gleichen Ordner befinden wie DUMPGEM.PRG bzw. DUMPGEM.EXE. In der Datei DUMPGEM.DAT befinden sich alle VDI-Befehle bis GEM-Version 3.11. Sie kann für spätere GEM-Versionen einfach mit einem Texteditor erweitert werden.

– Erweiterungen für Metadateien

Im folgenden sollen 2 Erweiterungen für Metadateien besprochen werden, die von einer Applikation aus in eine Metadatei hineingeschrieben werden können und Zusatzinformationen beinhalten. Dazu benutzen wir die Möglichkeit, eigene selbstdefinierte Subopcodes, die ab 101 beginnen müssen, in die Metadatei hineinzuschreiben. Halten sich alle Softwarehersteller an die Vereinbarung, wird der Datenaustausch wesentlich erleichtert.

Diese Zusatzinformationen, die in eine Metadatei geschrieben werden sollen, sind folgende:

1. Der Name und die Versionsnummer der Applikation, die die Metadatei erstellt hat.
2. Die Farbtabelle für die Farbindizes.

Die Vorteile sind klar: Kennt man die Versionsnummer und den Namen der Applikation, die die Metadatei erstellt hat, so kann man selbst bestimmen, ob man mit der Metadatei etwas anfangen will. Die Versionsnummer kann hilfreich sein, wenn Informationen in älteren Metadateien nicht mehr in neueren Applikationen unterstützt werden.

Die Farbtabelle gibt an, wie die Farben auf dem Gerät, auf dem die Metadatei erstellt wurde, eingestellt waren. Auf einem Rechner könnte man z.B. eine Grafik mit 16 Graustufen erstellen. Liest man die Grafik z.B. in ein Desktop-Publishing-System ein, so erscheint sie kunterbunt. Deshalb gibt man die RGB-Werte jeder Farbe an und schreibt sie in die Metadatei.

Für Name und Versionsnummer benötigen wir einen neuen Subopcode. Der neue Subopcode für die Funktion v_write_meta, der im Feld intin [0] angegeben wird, lautet also:

```
#define VM_VER_APP 101
```

a) Version und Applikation

Das sogenannte „Binding" der neuen Funktion lautet:

```
/*****************************************************************/
```

```
GLOBAL VOID vm_ver_app (vdi_handle, version, app_name)
WORD  vdi_handle, version;
UBYTE *app_name;

{
  WORD i;

  contrl [0] = 5;
  contrl [1] = 0;
  contrl [3] = 2 + strlen (app_name);
  contrl [5] = 99; /* v_write_meta */
  contrl [6] = vdi_handle;

  intin [0] = VM_VER_APP;
  intin [1] = version;

  i = 2;
  while ((intin [i++] = (WORD)(UBYTE)*app_name++) != 0);

  vdi ();
} /* vm_ver_app */

/*************************************************************************/
```

Beispiel: Die Versionsnummer 1.32 (GEM-Format = 100 * Hauptversionsnummer + Nebenversionsnummer) und der Name der Applikation (z.B. GEMDRAW) soll in die Metadatei geschrieben werden.

```
WORD version;
BYTE app_name [80];

version = 132;
strcpy (app_name, "GEMDRAW");
vm_ver_app (vdi_handle, version, app_name);
```

Achtung! Der Name der Applikation muß immer ohne Suffix angegeben werden, da sie sich auf dem ATARI ST und unter MS-DOS unterscheiden (z.B. TEST.PRG, TEST.APP).

b) Farbtabelle

Mit dem VDI-Befehl vs_color kann zu jedem Farbindex eine neue Farbmischung aus Rot-, Grün- und Blauwerten definiert werden. Ebenso kann die aktuelle Farbmischung eines Farbindexes mittels vq_color ausgelesen werden.

Beispiel: Aus einer laufenden Applikation heraus sollen die aktuellen Farbmischungen der Farbindizes in die Metadatei geschrieben werden. Jeder RGB-Wert liegt im Zahlenbereich 0 – 1000, wie dies bei der Benutzung von vq_color im VDI definiert ist.

```
WORD colors;
WORD rgb [3];
screen_handle = open_work (SCREEN); /* siehe SHOWGEM.C */
colors        = work_out [13];
meta_handle   = open_work (METAFILE);

for (i = 0; i < colors; i++)
{
  vq_color (screen_handle, i, FALSE, rgb);
  vs_color (meta_handle, i, rgb);
} /* for */
```

Ein letztes Wort zu Metadateien: Im Kopf befindet sich die Angabe der Seitenbreite und Seitenhöhe (pwidth und pheight). Möchte man Metadateien erstellen, so wird man diesen beiden Werten Standardgrößen geben. Folgende Standardgrößen werden von den meisten GEM-Programmen (z.B. GEM OUTPUT 3.11) unterstützt:

| Bezeichnung | Zoll | Zentimeter |
|---|---|---|
| Letter | 8,50 * 11,00 | 21,59 * 27,94 |
| Legal | 8,50 * 14,00 | 21,59 * 35,56 |
| Half | 8,50 * 5,50 | 21,59 * 13,97 |
| Ledger | 11,00 * 17,00 | 27,94 * 43,18 |
| Din A3 | 11,69 * 16,54 | 29,70 * 42,00 |
| Din A4 | 8,07 * 11,69 | 21,00 * 29,70 |
| Din A5 | 5,85 * 8,27 | 14,85 * 21,00 |
| Din B5 | 7,17 * 9,84 | 18,20 * 25,00 |
| Wide | 14,00 * 11,00 | 35,56 * 27,94 |

Die pwidth- und pheight-Werte ergeben sich aus den Zentimeter-Werten multipliziert mit 100.

### 2.7.2 Bit-Image-Format

Pixelgrafiken, im Gegensatz zu objektorientierten Vektorgrafiken, werden im GEM-System in einem Standardformat abgelegt. Dieses Standardformat nennt man Bit-Image-Format. Es stellt eine komprimierte Speicherung von Raster- oder Pixelbildern dar. Dateien, die solch eine Rastergrafik enthalten, haben immer das Suffix „.IMG".

Hauptsächlich gibt es 2 Vorteile, Pixelgrafiken im GEM-Standardformat abzuspeichern. Zum einen kann mit dem VDI-Befehl v_bit_image die komplette Grafik auf einen Drucker oder auch eine Kamera ausgegeben werden. Zum anderen können Pixelgrafiken zwischen verschiedenen Programmen ausgetauscht werden. Die Applikation WORD-PLUS, die zu ihren Anfangszeiten noch ein eigenes Grafikformat benutzte, hat jetzt um-

geschwenkt auf das GEM-Bit-Image-Format. Alle Anwender von GEM-Software auf dem ATARI ST würden es außerdem begrüßen, wenn die Grafikdaten endlich zwischen verschiedenen Applikationen austauschbar wären.

Bevor der Aufbau einer IMG-Datei beschrieben wird, wollen wir uns über die Zukunft Gedanken machen. Als das GEM-System 1984 herauskam, hatte ein PC mit EGA-Karte eine Auflösung von 640x350 Punkten und 16 gleichzeitig darstellbaren Farben. 5 Jahre später erreichten Grafikkarten Auflösungen von 1280x960 Punkten oder 16 Millionen Farben gleichzeitig. Das Bit-Image-Format ist zwar so allgemein gehalten, daß es beliebige Auflösungen und Farbenpracht darstellen kann, hat aber einen Nachteil.

Man kann aus einer IMG-Datei nicht herauslesen, wie die aktuelle Farbeinstellung war, als das Bild erzeugt wurde. Wer kann schon sagen, welche Farbe der Farbindex 183 hatte, als ein Bild mit 256 Farben digitalisiert abgespeichert wurde. War es eine Graustufe oder eine andere Farbe?

Glücklicherweise ist die Struktur einer Bit-Image-Datei so flexibel, daß beliebig viele Zusatzinformationen vor die reinen Pixeldaten geschrieben werden können. Der Kopf einer IMG-Datei wird von uns jetzt um die fehlenden Informationen erweitert und zwar so, daß das Format für alle Zeiten gerüstet ist. Das neue Format bleibt mit dem alten kompatibel und wird rechnerunabhängig gestaltet.

Bevor wir auf das neue Format eingehen, müssen wir verstehen, wie Farben abgelegt werden können. Dazu machen wir eine kleine Exkursion in die Technik der Farbmodelle.

Zur Zeit existieren 3 wichtige Farbmodelle sowie ein Farbsystem:

a) RGB-Modell
b) CMY-Modell
c) HLS-Modell
d) Pantone Farbsystem

a) Das RGB (Rot-Grün-Blau) Modell kennt fast jeder, denn die meisten Softwareentwickler können entweder auf einen Farbmonitor oder einen Farbfernseher zurückgreifen. Farben werden in diesem Modell additiv dargestellt. Jede Farbe stellt sich aus den 3 Anteilen Rot, Grün und Blau zusammen. Nähert man sich mit einer Lupe einem Farbbildschirm, so kann man die Schlitze dieser drei Farben erkennen.

Wenn wir annehmen, daß die Einzelwerte einer Farbkomponente (R,G,B) im Bereich 0 bis 1 liegen, so ergibt sich Schwarz zu (0,0,0) und Weiß zu (1,1,1). Grün hat dann den Wert (0,1,0). Alle Farben sind mit diesen 3 Werten darstellbar. Wieviele Farbtöne man auf einem Computersystem tatsächlich darstellen kann, hängt davon ab, wieviele Werte eine Einzelkomponente annehmen kann. Hat man z.B. 16 Werte pro Komponente, so ergeben sich

$$16 \times 16 \times 16 = 4096$$

verschiedene Farbtöne. Die Gesamtheit dieser Töne nennt man auch die Farbpalette. Wieviel Farbtöne letztlich auf einem Bildschirm dargestellt werden können, hängt von der Anzahl der Bits ab, die pro Bildschirmpunkt reserviert werden. Diese Zahl nennt man Anzahl der Farbebenen (planes). Benutzt man z.B. 4 Bits pro Bildschirmpunkt, so ergeben sich 16 verschiedene Farben, die sich gleichzeitig darstellen lassen. Diese Farben lassen sich dann meist aus der o.g. Farbpalette frei wählen. So könnte man dann ein Schwarzweißbild mit 16 Graustufen darstellen.

Im GEM-System hat man sich dafür entschieden, RGB-Werte von 0 – 1000 pro Farbton zuzulassen. Damit ergäben sich

$$1000 \times 1000 \times 1000 = 1000000000 = 1 \text{ Milliarde}$$

Farbtöne. Diese Anzahl kann das menschliche Auge nicht mehr unterscheiden, die Auflösung ist also ausreichend für alle Zeiten. Schwarz hat einen RGB-Wert von (0,0,0) und Weiß einen Wert von (1000,1000,1000).

b) Das CMY-Modell benutzt die Komplementärfarben des RGB-Modells. Diese sind Cyan (Türkis), Yellow (Gelb) und Magenta (Violett). Zwei Farben sind komplementär zueinander, wenn das Zusammenmischen die Farbe Weiß ergibt. So sind Cyan und Rot Komplementärfarben. Ein Gegenstand im Licht, der in der Farbe Cyan erscheint, absorbiert rot. Daher rührt auch der Name des subtraktiven Farbmodells.

c) Das HLS-Modell wird bei Macintosh-Software häufig benutzt und erhält seinen Namen durch die drei Begriffe Hue (Tönung), Lightness (Helligkeit) und Saturation (Sättigung). Jede Farbe kann durch die Angabe einer Winkelgröße und der Prozentzahlen für Helligkeit und Sättigung bestimmt werden. Rot liegt z.B. bei Winkel 120, Grün bei 220. Wir wollen hier nicht näher darauf eingehen, sondern nur die Existenz dieses Modells festhalten.

d) Da Farben auf Bildschirmen, Druckern und Belichtungsanlagen immer verschieden herauskommen, existiert in den USA das sogenannte Pantone Farbsystem. Dieses sind vordefinierte Farbpaletten. Zur Zeit kann man aus etwa 1000 Farbtönen wählen. Auch darauf soll hier nicht näher eingegangen werden.

Nachdem wir nun wissen, wie die Farben in einem Farbbild aufgebaut werden können, widmen wir uns der IMG-Datei. Sie besteht aus einem Kopf und den reinen Pixeldaten. Der Kopf, wie er von Digital Research definiert ist, lautet in C-Schreibweise:

```
typedef struct img_header
{
  WORD version;
  WORD headlen;
  WORD planes;
  WORD pat_run;
  WORD pix_width;
```

```
  WORD pix_height;
  WORD sl_width;
  WORD sl_height;
} IMG_HEADER;
```

Achtung! In einer Bit-Image-Datei liegt der Header immer im Motorola 68000-Format vor. Auf INTEL-Rechnern (MS-DOS, FLEXOS) muß er also byteweise gedreht werden.

version : Eine Versionsnummer, meist 1
headlen : Länge des Kopfes in 16-Bit-Worten (hier 8)
planes : Anzahl der Farbebenen, die die Grafik enthält
pat_run : Anzahl Bytes für einen „pattern run", (s.u.)
pix_width : Pixelbreite des Quellgerätes in 1/1000 mm
pix_height : Pixelhöhe des Quellgerätes in 1/1000 mm
sl_width : Breite einer „Scan-Zeile" in Pixel
sl_height : Höhe einer „Scan-Zeile" in Pixel

Möchte man eine Bit-Image-Datei erstellen, so kann man einige Werte, die man in den Kopf eintragen muß, leicht bestimmen. Addiert man den Wert headlen zum Anfang der Datei, so kommt man zu den reinen Pixeldaten. Die Anzahl der Farbebenen erhält man über die VDI-Funktion vq_extnd (work_out [4]). Der „pattern run" ist bei allen Bildschirmen 2, wobei die Werte pix_width und pix_height aus der Funktion v_opnvwk (work_out [3] – [4]) bestimmt werden können. Die Breite einer Scan-Zeile ist immer ein Vielfaches von 16. Bei einer Grafik mit 20 Pixel in horizontaler und 10 Pixel in vertikaler Richtung müssen die beiden sl-Werte wie folgt gesetzt werden:

```
sl_width  = 32;
sl_height = 10;
```

Ab der relativen Dateiposition „headlen * 2" beginnen die Pixeldaten einer Bit-Image-Datei. Sie sind komprimiert abgelegt, aber recht einfach wieder zu entpacken.

Eine komprimierte Grafik besteht aus sogenannten Scan-Zeilen. Sie geben die Pixelhöhe der Grafik an. Jeder Scan-Zeile geht ein Wiederholungsfaktor (VRC = vertical replication count) voraus, der angibt, wie oft diese Scan-Zeile wiederholt werden soll. Sind also n Zeilen einer Grafik gleich, so ist der Wiederholungsfaktor gleich n.

Um den Wiederholungsfaktor zu erkennen, wird er durch zwei 0-Bytes sowie dem Wert 255 (hexadezimal = 0xFF) eingeleitet, also:

| Byte | Wert |
|---|---|
| 0 | 0 |
| 1 | 0 |
| 2 | 0xFF |
| 3 | VRC |

Anschließend folgen die komprimierten Daten der Pixel. Dabei werden nacheinander alle Farbebenen beschrieben. Zuerst kommen also die Daten einer Scan-Zeile für Ebene 1, dann für Ebene 2 usw. Die Anzahl der Farbebenen, die in der Grafik benutzt werden, kann aus dem Kopf (planes) abgelesen werden.

Um eine Scan-Zeile günstig zu packen, werden 3 Methoden verwendet. Sie werden „solid run", „pattern run" und „bit string" genannt.

a) solid run

Wie der Name schon sagt, werden hiermit einfarbig gefüllte Flächen beschrieben. Ein solid run besteht aus einem einzigen Byte, wobei das höchste Bit angibt, ob der Punkt gesetzt ist oder nicht. Die unteren sieben Bit werden für die Länge L des run benutzt. Pro solid run werden 8 * L Punkte beschrieben.

Beispiel:

81: Linie mit 8 gesetzten Punkten
8A: Linie mit 80 gesetzten Punkten
08: Linie mit 64 nicht gesetzten Punkten

b) pattern run

Der pattern run dient zur Beschreibung von regelmäßigen Mustern. Diese Muster wiederholen sich auf Bildschirmgrafiken häufig alle 16 Bit, so daß in diesem Fall die Länge des Musters aus 2 Bytes besteht. Der Wert pat_run im Kopf der Datei gibt an, aus wievielen Bytes ein Muster besteht.

Der pattern run beginnt mit dem Wert 0, ihm folgen die Anzahl der Musterwiederholungen (n) und die Musterbeschreibung (meist 2 Bytes):

| Byte | Wert |
|---|---|
| 0 | 0 |
| 1 | n |
| 2 | erstes Byte des Musters |
| . | |
| . | |
| pat_run + 1 | letztes Byte des Musters |

Beispiel: Folgendes Muster soll sich 5-mal wiederholen:

''****  ****  '' (2-mal 4 Punkte und 4 Leerzeichen)

Es wird dann komprimiert zu den vier Bytes: 00 05 F0 F0

c) bit string

In den Fällen, wo kein solid run oder kein pattern run angewendet werden kann, werden die Bits direkt gespeichert. Ein bit string hat folgendes Aussehen:

| Byte | Wert |
|---|---|
| 0 | 0x80 |
| 1 | n |
| 2 | erstes Byte |
| . | |
| . | |
| n+1 | letztes Byte |

Beispiel: Folgendes Muster soll als bit string kodiert werden:

''** * *** ******''

Die Bytefolge sieht dann so aus: 80 02 D7 3F

Die Reihenfolge des Dekodierens ist wichtig, da sonst die Methoden des Packens nicht erkannt werden. Pixeldaten müssen also z.B. wie folgt dekodiert werden (Algorithmus):

1. Ist Byte = 0, dann schaue auf Byte + 1.
   Ist Byte + 1 = 0,
   dann vertical replication count,
   sonst pattern run.
2. Ist Byte = 0x80, so bit string.
3. Keine der obigen Fälle, dann solid run.

Achtung: Der Wert sl_width (Breite einer Scan-Zeile in Pixels) kann bis zu 7 Bit kleiner sein, als die Daten, die eine Scan-Zeile beschreibt, da Scan-Zeilen byteweise beschrieben werden (siehe auch Algorithmus in SHOWGEM.C). Ist z.B. sl_width = 1, so werden trotzdem 8 Bit pro Scan-Zeile beschrieben, also 7 Bit zuviel. Erzeugt man eine Grafik mit GEMPAINT, so ist der Wert sl_width immer eine durch 16 teilbare Zahl. Hier werden also immer mindestens 16 Bit beschrieben, auch wenn nur ein einziges Pixel (das am weitestens links stehende) gesetzt ist. Außerdem erzeugt GEMPAINT zu jeder IMG-Datei eine zugehörige Datei mit Suffix „.GEM". Sie enthält Zusatzinformationen, wie Größe der Seite und Ausdehnung der darin enthaltenen Grafik und den Dateinamen der Image-Datei. Schauen Sie sich die entsprechenden Dateien mit DUMPGEM einmal genauer an.

Zum Schluß noch ein allgemeines Wort über die Datendarstellung. Wie man an obigen Beispielen sieht, werden die einzelnen Bits immer im Standardformat abgelegt. D.h., daß das höchstwertige Bit immer den am weitesten links stehenden Punkt beschreibt. Auf der Beispieldiskette befindet sich die Datei TIGER.IMG und COLORS.IMG. Ersteres ist ein monochromes Bild, letzteres eine Grafik mit 16 Farben, also 4 Farbebenen.

– Das neue Bit-Image-Format

Im neuen Format ist es unser Ziel, auch die Farbtabelle (RGB-Werte pro Farbindex) im Kopf unterzubringen. Der RGB-Wert jeder Farbe soll dabei geräteunabhängig beschrieben werden. Wir nennen die Beschreibungsmöglichkeiten gemäß dem VDI auch Standardformat.

Gehen wir von einem bestimmten Grafikformat aus, so interessiert noch, wieviele Werte jede einzelne Farbe auf dem Quellgerät maximal annehmen konnte. Bei einem ATARI ST ist diese Zahl z.B. 8, da jeder RGB-Wert acht unterschiedliche Einstellungen annehmen kann. Aus diesem Wert kann man auch den Wert für die Standardeinstellung bestimmen. VDI-Standardeinstellungen liegen im Bereich von 0 bis 1000. Für 8 unterschiedliche RGB-Werte heißt dies:

| Spezifisch | Standard |
|---|---|
| 0 | 0 |
| 1 | 142 |
| 2 | 285 |
| 3 | 428 |
| 4 | 571 |
| 5 | 714 |
| 6 | 857 |
| 7 | 1000 |

Um die Farbtabelle darzustellen, benutzen wir für jeden Eintrag einen 48-Bit-Wert, der die RGB-Werte beschreibt. D.h., daß für jede Farbkomponente Einstellungen von 0 – 65535 vorgenommen werden können. Daraus ergibt sich der Bereich, aus denen die Farben gewählt werden können zu:

```
2^16 * 2^16 * 2^16 = 2^48
```

Die ergibt etwa eine Auswahl aus 300 Billionen verschiedener Farben. Da es sich bei diesen Werten um RGB- oder CMY-Werte oder andere Werte handeln kann, legen wir das Farbmodell ebenfalls im erweiterten Kopf ab. Der Wert color_model kann 0 (RGB), 1 (CYM), 2 (HLS) oder 3 (Pantone) sein. Im Fall „Pantone" beschreibt der Rot-Wert die Pantone-Farbe.

Falls Digital Research den Kopf erweitern sollte, setzen wir noch einen Erkennungswert (x_id) für unseren erweiterten Kopf an die erste Stelle hinter dem alten Kopf. Dieser Wert soll die Zeichenkette „XIMG" enthalten.

Damit wird die Beschreibung des erweiterten Kopfes (XIMG_HEADER) wie folgt festgelegt:

```
#define RGB     0
#define CYM     1
#define HLS     2
#define PANTONE 3

typedef struct
{
  UWORD red;
  UWORD green;
  UWORD blue;
} RGB_LIST;

typedef struct ximg_header
{
  WORD      version;
  WORD      headlen;
  WORD      planes;
  WORD      pat_run;
  WORD      pix_width;
  WORD      pix_height;
  WORD      sl_width;
  WORD      sl_height;
  BYTE      x_id [4];
  WORD      color_model;   /* 0 = RGB,1 = CYM,2 = HLS,3 = Pantone */
  RGB_LIST  color_table[]; /* open array, length 2^planes */
} XIMG_HEADER;
```

Der gesamte Kopf liegt natürlich im 68000-Format vor, d.h. auf INTEL-Rechner muß er wort- bzw. langwortweise gedreht werden. Das Drehen von Wörtern und Langwörtern wurde weiter oben im Programm GRC (flip_long, flip_word) gezeigt.

Zum erweiterten Kopf ein Beispiel: Auf einem Rechner wurde eine Pixelgrafik der Größe $32 \times 20$ Pixel erstellt. Der Bildschirm konnte 16 Farben (4 Farbebenen) aus einer Palette von 512 (8 Stufen pro RGB-Wert) gleichzeitig darstellen. Es wurde das RGB-Modell gewählt. Der erweiterte Kopf muß dann wie folgt aussehen:

```
#define COLORS 16;  /* 16 = 2^planes */

RGB_LIST    rgb_list [COLORS];
XIMG_HEADER x_header;

x_header.version     = 1;
x_header.headlen     = sizeof (XIMG_HEADER) + sizeof (rgb_list);
x_header.planes      = 4;
x_header.pat_run     = 2;
x_header.pix_width   = work_out [3]; /* nach opnvwk */
x_header.pix_height  = work_out [4];
```

```
x_header.sl_width    = 32;
x_header.sl_height   = 20;
strcpy (x_id, "XIMG");
x_header.color_model = RGB;
```

Das Feld rgb_list wurde vorher natürlich mit den entsprechenden Werten gefüllt. Der Kopf und die Farbtabelle kann nun nach eventuellem Drehen der Daten auf eine Datei geschrieben werden:

```
#if I8068
  flip_word ((BYTE *)&x_header.version);
  flip_word ((BYTE *)&x_header.headlen);
  flip_word ((BYTE *)&x_header.planes);
  flip_word ((BYTE *)&x_header.pat_run);
  flip_word ((BYTE *)&x_header.pix_width);
  flip_word ((BYTE *)&x_header.pix_height);
  flip_word ((BYTE *)&x_header.sl_width);
  flip_word ((BYTE *)&x_header.sl_height);
  flip_word ((BYTE *)&x_header.color_model);

  for (i = 0; i < COLORS; i++)
  {
    flip_word ((BYTE *)&rgb_list [i].red);
    flip_word ((BYTE *)&rgb_list [i].green);
    flip_word ((BYTE *)&rgb_list [i].blue);
  } /* for */
#endif

handle = Fcreate ("GRAFIK.IMG", 0);
Fwrite (handle, (LONG)sizeof (XIMG_HEADER), &x_header);
Fwrite (handle, (LONG)sizeof (rgb_list), rgb_list);
... jetzt gepackte Grafikdaten schreiben...
Fclose (handle);
```

Entsprechend kann die Bit-Image-Datei wieder eingelesen werden (Programmfragment):

```
XIMG_HEDAER *x_header;
UBYTE       *buffer;
LONG        size;
WORD        i, colors;

handle  = Fopen ("GRAFIK.IMG", 0);
size    = Fseek (0L, handle, 2); /* Dateigröße */
buffer  = Malloc (size);
Fread (handle, size, buffer);
x_header = (XIMG_HEADER *)buffer;
```

```
#if I8086
  flip... (Kopf drehen, siehe oben)
#endif

colors = 2 << x_header->planes; /* 2^planes */

/* Farbtabelle auf Bildschirm ausgeben */
for (i = 0; i < colors; i++)
  printf ("rgb [%2d] = %3d,%3d,%3d\n",
          i,
          x_header->color_table [i].red,
          x_header->color_table [i].green,
          x_header->color_table [i].blue);
```

### 2.7.3 OUT-Dateiformat

Wie bei der Funktion v_alpha_text in Kapitel 2.3.7 beschrieben, können Steuerzeichen für Textattribute in eine Datei mit Suffix „.OUT" geschrieben werden. Der entsprechende Ausgabetreiber (z.B. für den Drucker) erkennt diese Steuerzeichen und schaltet das Ausgabegerät in den entsprechenden Modus um.

Sollen Textdaten mit Attributen (fett, kursiv, etc.) zwischen verschiedenen Applikationen ausgetauscht werden, so empfehlen wir das von Digital Research standardisierte OUT-Dateiformat.

Es ist eine ASCII-Datei, die zum Umschalten zwischen Textattributen den Steuercode DC2 (ASCII-Wert 18) benutzt. Danch folgt ein Buchstabe, der das Ein- oder Ausschalten des Attributes beschreibt.

(DC2)0 – Fettschrift ein
(DC2)1 – Fettschrift aus
(DC2)2 – Kursivschrift ein
(DC2)3 – Kursivschrift aus
(DC2)4 – Unterstreichen ein
(DC2)5 – Unterstreichen aus
(DC2)6 – Superscript ein
(DC2)7 – Superscript aus
(DC2)8 – Subscript ein
(DC2)9 – Subscript aus
(DC2)A – NLQ-Schrift ein
(DC2)B – NLQ-Schrift aus
(DC2)C – Breitschrift ein
(DC2)D – Breitschrift aus
(DC2)E – Hellschrift ein

(DC2)F – Hellschrift aus
(DC2)G – (DC2)V – reserviert
(DC2)W – Pica ein
(DC2)X – Elite ein
(DC2)Y – Schmalschrift ein
(DC2)Z – Proportionalschrift ein

Der Text „Dies ist **fett** bzw. *kursiv*." müßte dann so aussehen:

Dies ist (DC2)0fett(DC2)1 bzw. (DC2)2kursiv(DC2)3.

Außerdem besteht die Möglichkeit, Grafiken mit in die Ausgabedatei einzubinden. Dabei wird eine Einleitungssequenz von 2 Escape-Zeichen benutzt. Die Zeile ist wie folgt aufgebaut:

(ESC)(ESC)GEM,x,y,w,h,D:\PATHNAME\FILENAME.EXT

Die Werte x, y, w und h geben ein Rechteck relativ zur aktuellen Cursorposition des Ausgabegerätes an. Die Einheit wird in Zeichenzellen angegeben. „D:" ist der Laufwerksname, auf dem sich die Grafik befindet. Das Bild TEST.IMG auf Laufwerk C im Ordner \GEMAPPS\IMAGES, das sich 2 Zeichen rechts und 1 Zeile unter dem Text „Hier ist das Bild" befindet und 60 Zeichen breit sowie 10 Zeilen hoch sein soll, muß wie folgt kodiert werden:

Hier ist das Bild(CR)(LF)
(CR)(LF)
(ESC)(ESC)GEM,2,0,60,10,C:\GEMAPPS\IMAGES\TEST.IMG(CR)(LF)

Der Druckertreiber versteht diese Sequenzen und gibt das gewünschte Ergebnis auf dem Ausgabegerät aus.

# 3 Entwicklungsumgebungen

Als Verfechter der „sauberen" Programmierung können wir nur allen Lesern dieses Buches raten, sich am Programmierstil der vorgestellten Programme ein Beispiel zu nehmen. Wir legen den allergrößten Wert auf Portabilität. Den Preis, den man dafür bezahlen muß, ist äußerst gering: Lediglich ein paar Programmzeilen mehr müssen geschrieben werden (siehe VDITEST, GRC, RCM). Mit einigen „defines" der Form

```
#if GEMDOS
...
#endif
```

kann der gleiche Quelltext auf allen Rechnern übersetzt werden.

Das in Kapitel 6 vorgestellte Programm SCRAP ist bis heute das einzige Programm, das sich auf jedem GEM-System ohne Fehler und Modifikationen übersetzen läßt und problemlos der Umgebung anpaßt.

Eine traurige Ausnahme bildet allerdings das Programm SHOWGEM auf der beiliegenden Diskette. Unter FlexOS und X/GEM werden VDI-Aufrufe leider nicht mehr über die contrl-, intin- und ptsin-Felder getätigt. Um Metadateien auszugeben, bedarf es eines Interpreters, der aus den Informationen in den o.g. Feldern „echte" VDI-Aufrufe macht. Da wir bisher keine Zeit für solch einen Interpreter hatten, muß der interessierte Leser diesen selbst bauen. Alle Informationen über Metadateien wurden ja in den vorangegangenen Kapiteln dargelegt, so daß die Aufgabe nicht allzu schwer fallen dürfte.

a) Portabilität

Einige werden sich fragen: „Wozu Portabilität"? Dies ist richtig, wenn man nur auf einen Rechner und auf ein Betriebssystem fixiert ist. Spätestens dann, wenn es ums Vermarkten von Applikationen geht, sieht die Sache anders aus. Stellen Sie sich vor, Sie schreiben eine tolle Tabellenkalkulation auf dem ATARI ST und möchten diese vertreiben. Sie merken aber, daß noch mehr Kunden daran interessiert sind, die nicht aus ATARI-Kreisen stammen; z.B. fehlen unter X/GEM noch sehr viele GEM-Applikationen, da noch nichts angepaßt wurde. Wenn Sie jetzt Ihre Applikation mit den Hilfsmitteln aus diesem Buch

geschrieben haben, so können Sie am gleichen Tag, an dem die Programmierung auf dem ATARI ST abgeschlossen ist, z.B. bei Siemens anrufen und dort Ihre Applikation für X/GEM verkaufen. Lediglich ein Übersetzungslauf von einigen Minuten ist nötig.

Portabilität ist aber nicht nur auf verschiedene Rechner und Betriebssysteme fixiert, sondern ist ebenso wichtig für eine bestimmte GEM-Version. Applikationen, die auf GEM 1.X laufen, können ohne Probleme auf neueren Versionen ausgeführt werden.

Wenn vorhin von „sauberer" Programmierung gesprochen wurde, so ist damit auch gemeint, daß keine hardwareabhängigen Funktionen benutzt werden sollten. Versucht man z.B., ein Programm für ATARI ST auf einem Großbildschirm mit $1280 \times 960$ Punkten laufen zu lassen, so erlebt man fast nur Programmabstürze. Für den ATARI TT gilt das gleiche. Einige werden zwar jetzt behaupten, daß dadurch die Performance verloren ginge, aber das ist bei den meisten Programmen nicht so wichtig. Wenn einem Softwareentwickler dies dennoch wichtig erscheint, so soll er den Algorithmus zunächst in der Sprache C formulieren und einbinden. Läuft er, so sollte er aus Portabilitätsgründen gesichert werden. Anschließend kann er den Algorithmus in Assembler schreiben und optimieren. So bleibt er portabel für andere Betriebssysteme oder andere Hardware, und das Programm läuft trotzdem schnell.

Portabilität bezieht sich auch auf die verwendeten Compiler. So benutzt der Lattice C-Compiler als int-Werte immer 32 Bit. Auch werden immer 32 Bit für jeden Parameter auf den Stack geschrieben. Hier empfiehlt sich die Verwendung eigener Datentypen. So könnte ein 16-Bit-Wort für Lattice und andere Compiler wie folgt beschrieben werden:

```
#if LATTICE
#define WORD short
#else
#define WORD int
#endif
```

Die 16-Bit-Variable i kann dann als

```
WORD i;
```

beschrieben werden. Programme werden also portabel bezüglich der verwendeten Datentypen.

Portabilität bezüglich der VDI- und AES-Funktionen ist ebenfalls ein Thema für sich. Während unter MS-DOS die Quellen der sogenannten Bindings aller Funktionen im GEM-Programmer's-Toolkit mitgeliefert werden, sind Entwickler auf einem ATARI ST von den Compilerherstellern abhängig. Dort werden zum Compiler nur die fertigen VDI- und AES-Libraries mitgeliefert. Diese entsprechen leider nicht immer der Namensgebung aus dem Original-Toolkit von Digital Research. So wird die AES-Funktion graf_mbox häufig falsch als graf_movebox in die AES-Library gepackt. Die Funktion graf_movebox aber existiert nicht unter MS-DOS, Programme sind also nicht mehr portabel.

So kann man wieder zu einem Trick greifen. Z.B. wird für TURBO C auf dem ATARI ST die Funktion wie folgt umdefiniert:

```
#if GEMDOS
#if TURBO_C
#define graf_mbox graf_movebox
#endif
#endif
```

Der Programmierer schreibt in sein Progamm immer graf_mbox und der Präprozessor des benutzten Compilers ersetzt diese Funktion durch den Aufruf graf_movebox, was wiederum dem Linker gefällt, da er diese Funktion jetzt aus der (falschen) AES-Library holen kann.

b) ANSI-Standard

Neuere Compiler werden nach der ANSI-Norm entwickelt. Durch die neue ANSI-Norm ergeben sich in C-Programmen neue Möglichkeiten, und es werden Fehler verhindert. Das größte Manko an C war bisher, daß keine Typüberprüfung der Parameter bei Funktionen getätigt wurde. Dies hatte zur Folge, daß auf Parameter innerhalb einer Funktion falsch zugegriffen wurde, wenn ein anderer Datentyp übergeben wurde. Beispiel:

```
#include <stdio.h>
#include <portab.h>

/*****************************************/

GLOBAL WORD main ()

{
  LONG s;

  s = summe (1, 2);
  printf ("Summe: %ld\n", s);

  return (0);
} /* main */

/*****************************************/

LOCAL LONG summe (a, b)
LONG a, b;

{
   return (a + b);
} summe /*summe*/

/*****************************************/
```

Die Funktion Summe soll die Summe zweier LONG-Werte berechnen und zurückliefern. Das obige Programm wird nicht korrekt laufen, da die beiden übergebenen Zahlen vom Compiler als 16-Bit-Integer-Wert (int) interpretiert werden. So werden zwei 16-Bit-Werte übergeben, wobei die Funktion Summe aber auf 32-Bit-(LONG)-Werte zugreift.

Man hätte dies verhindern können, indem man die Summe wie folgt berechnet hätte:

```
s = summe (1L, 2L); oder
s = summe ((LONG)1, (LONG)2);
```

Im ersten Fall werden die beiden Zahlen 1 und 2 als Langwerte übergeben (Suffix L). Im zweiten Fall werden die beiden Zahlen durch die sogenannte Cast-Operation auf den Datentyp LONG umgewandelt.

Aber da Menschen Fehler machen, passiert es häufig, daß man an eine Funktion falsche Datentypen übergibt. Was in Pascal und Modula schon lange geht, sollte auch in ANSI-Compilern möglich sein. Dort werden die übergebenen Datentypen nicht nur geprüft, sondern gegebenfalls automatisch an die Funktion angepaßt. Auch Typüberprüfung bei Zuweisungen (Typ der linken Seite = Typ der rechten Seite) werden vorgenommen und eine Warnung gebracht, wenn die Typen nicht übereinstimmen. In unseren Beispielprogrammen wird deswegen auch häufig die Cast-Operation angewendet, um Warnungen des Compilers zu vermeiden.

Die Typanpassung bei Funktionsaufrufen kann aber nur vorgenommen werden, wenn die Funktion vorher bekannt ist. Deshalb ist man dazu übergegangen, alle Funktionen, die in einem Modul vorkommen, durch eine forward-Deklaration mit Datentypen bekannt zu machen. Das obige Beispiel sieht dann so aus:

```
#include <stdio.h>
#include <portab.h>

/*****************************************/

LOCAL LONG summe (LONG a, LONG b);

/*****************************************/

GLOBAL WORD main ()

{
  LONG s;

  s = summe (1, 2);
  printf ("Summe: %ld\n", s);

  return (0);
```

```
} /* main */

/*************************************/

LOCAL LONG summe (a, b)
LONG a, b;

{
   return (a + b);
} /* summe */

/*************************************/
```

Jetzt weiß der Compiler zur Übersetzungszeit, daß die beiden Parameter Langwerte sein müssen, und paßt die beiden Zahlen 1 und 2 automatisch an, die dann an die Funktion übergeben werden.

Was machen aber Entwickler, die noch keinen ANSI-Compiler zur Verfügung haben? Sie können das obige Programm so nicht mehr übersetzen lassen. Aber auch für sie gibt es einen Trick. Dazu bedienen wir uns der Möglichkeit des Präprozessors, Wörter oder ganze Ausdrücke als „nichts" zu definieren.

Beispiel: Das Wort CDECL, das in manchen Compilern existiert, soll entsprechend gelöscht werden:

```
#if TURBO_C
#define CDECL cdecl
#else
#define CDECL
#endif
```

Die Funktionsdefinition

```
LOCAL WORD CDECL name (x)
WORD x;
{
}
```

wird im Turbo-C-Compiler, der das Wort cdecl kennt, wie folgt umgewandelt:

```
static int cdecl name (x)
int x;
{
}
```

In allen anderen Compilern verwandelt der Präprozessor die Funktionsdefinition in:

```
static int name (x)
int x;
{
}
```

Das Wort cdecl wurde durch die Leerdefinition „#define CDECL" sozusagen wegdefiniert.

Beim Problem unserer Prototypen hilft dieser Trick genauso. Dort verwenden wir für ANSI-Compiler ein anderes Makro als für Compiler, die den ANSI-Standard nicht kennen. Wir definieren folgende Makros:

```
#if MS_C ¦ TURBO_C ¦ HIGH_C    /* ANSI compilers */
#define ANSI 1
#define _(params) params       /* parameter checking */
#else
#define ANSI 0
#define _(params) ()           /* no parameter checking */
#endif
```

ANSI-Compiler sind Microsoft C (MS_C), Turbo C (TURBO_C) und High C (HIGH_C). Alle anderen Compiler wie Lattice C, Mark Williams C usw. kennen den ANSI-Standard nicht.

Jetzt können wir Prototypen von Funktionen wie folgt definieren:

```
LOCAL LONG summe _((LONG a, LONG b));
```

Bei ANSI-Compilern werden durch die Makrodefinition die beiden äußeren Klammern sowie der Unterstrich durch das Innere der äußeren Klammern ersetzt. Der Präprozessor macht also daraus:

```
static long summe (long a, long b);
```

Der Präprozessor von Compilern ohne ANSI-Standard ersetzt das Innere der äußeren Klammern durch „nichts". Die Deklaration sieht also wie folgt aus:

```
static long summe ();
```

Diese Deklaration entspricht der „normalen" forward-Deklaration von Funktionen, die einen anderen Wert als „int" zurückgeben. Wer also seine Funktionen mit den Prototypen wie oben gezeigt definiert, ist über alle Compiler hinweg portabel. ANSI oder nicht spielt dann keine Rolle mehr.

Im folgenden wird zu den o.g. Problemen die Include-Datei PORTAB.H vorgestellt. Sie enthält alle Unterschiede (sprachliche und GEM-spezifische) der folgenden Betriebssysteme, Prozessoren, Compiler und GEM-Versionen:

a) Bertriebssysteme:

- GEMDOS (Digital Research, auf ATARI ST verwendet)
- MS-DOS (Microsoft auf IBM-PC's und Kompatiblen)
- OS/2 (Microsoft auf IBM-PC's und Kompatiblen ab 80286)
- FlexOS (Digital Research, auf Siemens Programmier-Geräten)

b) Prozessoren

- MOTOROLA (MOTOROLA Serie: 68000, 68010, 68020, 68030, 68040)
- INTEL (INTEL Serie: 8088, 8086, 80186, 80286, 80386, 80486)

c) Compiler

- Digital Research C (Digital Research Inc.)
- LASER C (Megamax Inc.)
- LATTICE C (Metacomco)
- MARK WILLIAMS C (Mark Williams Company)
- TURBO C (Borland, Heimsoeth)
- MICROSOFT C (Microsoft)
- HIGH C (Metaware Inc.)

d) GEM-Versionen

- GEM 1.X (ATARI ST, ältere PC-Version)
- GEM 2.X (PC, ATARI ST-Version von ABC-Software)
- GEM 3.X (aktuelle PC-Version GEM 3.01 und GEM 3.11)
- X/GEM (Multitasking-Version unter FlexOS und OS/2)

```
/*****************************************************************/
/*                                                               */
/* PORTAB.H                                                      */
/*                                                               */
/* Use of this file may make your code compatible with           */
/* all C compilers listed.                                       */
/*                                                               */
/*****************************************************************/

/*****************************************************************/
/* ENVIRONMENT                                                   */
/*****************************************************************/
```

```
#ifndef __PORTAB__
#define __PORTAB__

#define GEMDOS      1               /* Digital Research GEMDOS      */
#define MSDOS       0               /* Microsoft MS-DOS             */
#define OS2         0               /* Microsoft OS/2               */
#define FLEXOS      0               /* Digital Research FlexOS      */

#define M68000      1               /* Motorola Processing Unit     */
#define I8086       0               /* Intel Processing Unit        */

#define DR_C        0               /* Digital Research C Compiler */
#define LASER_C     0               /* Laser C Compiler             */
#define LATTICE_C   0               /* Lattice C Compiler           */
#define MW_C        0               /* Mark Williams C Compiler     */
#define TURBO_C     1               /* Turbo C Compiler             */
#define MS_C        0               /* Microsoft C Compiler         */
#define HIGH_C      0               /* Metaware High C Compiler     */

#define GEM1        0x0001          /* ATARI GEM version            */
#define GEM2        0x0002          /* MS-DOS GEM 2.x versions      */
#define GEM3        0x0004          /* MS-DOS GEM/3 version         */
#define XGEM        0x0100          /* OS/2, FlexOS X/GEM version   */

#ifndef GEM
#if GEMDOS
#define GEM         GEM1            /* GEMDOS default is GEM1       */
#endif /* GEMDOS */

#if MSDOS
#define GEM         GEM3            /* MS-DOS default is GEM3       */
#endif /* MSDOS */

#if OS2
#define GEM         XGEM            /* OS/2 default is X/GEM        */
#endif /* MSDOS */

#if FLEXOS
#define GEM         XGEM            /* FlexOS default is X/GEM      */
#endif /* FLEXOS */
#endif /* GEM */

/*****************************************************************/
/* STANDARD TYPE DEFINITIONS                                     */
/*****************************************************************/
```

```
#define BYTE    char                /* Signed byte                  */
#define UBYTE   unsigned char       /* Unsigned byte                */

#if LATTICE_C
#define WORD    short               /* Signed word (16 bits)        */
#define UWORD   unsigned short      /* Unsigned word                */
#else
#define WORD    int                 /* Signed word (16 bits)        */
#define UWORD   unsigned int        /* Unsigned word                */
#endif

#define LONG    long                /* Signed long (32 bits)        */
#define ULONG   unsigned long       /* Unsigned long                */

#define BOOLEAN WORD                /* 2 valued (true/false)        */

#define FLOAT   float               /* Single precision float       */
#define DOUBLE  double              /* Double precision float       */

#define INT     int                 /* A machine dependent int      */
#define UINT    unsigned int        /* A machine dependent uint     */

#define REG     register            /* Register variable            */
#define AUTO    auto                /* Local to function            */
#define EXTERN  extern              /* External variable            */
#define LOCAL   static              /* Local to module              */
#define MLOCAL  LOCAL               /* Local to module              */
#define GLOBAL                      /* Global variable              */

/*****************************************************************/
/* COMPILER DEPENDENT DEFINITIONS                                */
/*****************************************************************/

#if GEMDOS                          /* GEMDOS compilers             */
#if DR_C
#define void WORD                   /* DR_C doesn't know void       */
#endif /* DR_C */

#if LASER_C
#define graf_mbox graf_movebox      /* Wrong GEM binding            */
#define graf_rubbox graf_rubberbox  /* Wrong GEM binding            */
#endif /* LASER_C */

#if LATTICE_C
#define graf_mbox graf_movebox      /* Wrong GEM binding            */
#define graf_rubbox graf_rubberbox  /* Wrong GEM binding            */
#endif /* LATTICE_C */
```

```
#if TURBO_C
#define graf_mbox graf_movebox      /* Wrong GEM binding          */
#define graf_rubbox graf_rubberbox  /* Wrong GEM binding          */
#endif /* TURBO_C */

#if MW_C
#define VOID WORD                   /* MW_C doesn't know (void *) */
#endif /* MW_C */

#if LATTICE_C
#define ADR(A) (LONG)A >> 16, (LONG)A & 0xFFFF
#else
#define ADR(A) (WORD)((LONG)A >> 16), (WORD)((LONG)A & 0xFFFF)
#endif /* LATTICE_C */
#endif /* GEMDOS */

#if MSDOS | OS2                     /* MS-DOS or OS2 compilers    */
#define ADR(A) (WORD)((LONG)A & 0xFFFF), (WORD)((LONG)A >> 16)
#endif /* MSDOS */

#if FLEXOS                          /* FlexOS compilers           */
#define ADR(A) (WORD)((LONG)A & 0xFFFF), (WORD)((LONG)A >> 16)
#endif /* FLEXOS */

#if MS_C | TURBO_C | HIGH_C         /* ANSI compilers             */
#define ANSI 1
#define _(params) params            /* parameter checking         */
#else
#define ANSI 0
#define _(params) ()                /* no parameter checking      */
#define const
#define volatile
#define size_t UINT
#endif

#if DR_C | LASER_C | LATTICE_C | MW_C | HIGH_C
#define cdecl
#define pascal
#endif

#define CONST    const
#define VOLATILE volatile
#define CDECL    cdecl
#define PASCAL   pascal
```

```
#define SIZE_T   size_t

#ifndef VOID
#define VOID     void
#endif

/*********************************************************************/
/* OPERATING SYSTEM DEPENDENT DEFINITIONS                            */
/*********************************************************************/

#if GEMDOS
#define FAR                         /* Far Pointer                   */
#define NEAR                        /* Near Pointer                  */
#else
#define FAR     far                 /* Far Pointer                   */
#define NEAR    near                /* Near Pointer                  */
#endif /* GEMDOS */

#if GEM & GEM1
#define appl_bvset(bvdisk, bvhard)
#define appl_yield() evnt_timer (0, 0)
#define menu_unregister(mid)
#define scrp_clear()
#define xgrf_stepcalc(orgw, orgh, xc, yc, w, h, pcx, pcy, pcnt,
                      pxstep, pystep)
#define xgrf_2box(xc, yc, w, h, corners, cnt, xstep, ystep, doubled)
#endif /* GEM1 */

#if GEM & (GEM1 | XGEM)
#define shel_rdef(lpcmd, lpdir)
#define shel_wdef(lpcmd, lpdir)
#endif /* GEM1 | XGEM */

#if GEM & (GEM1 | GEM2)
#define menu_click(click, setit)
#endif /* GEM1 | GEM2 */

#if GEM & (GEM2 | GEM3 | XGEM)
#define fsel_exinput(pipath, pisel, pbutton, plabel)\
        fsel_input  (pipath, pisel, pbutton)
#define wind_new()
#endif /* GEM2 | GEM3 | XGEM */

/*********************************************************************/
/* MISCELLANEOUS DEFINITIONS                                         */
/*********************************************************************/
```

```
#define FALSE   (BOOLEAN)0          /* Function FALSE value        */
#define TRUE    (BOOLEAN)1          /* Function TRUE  value        */
#define FAILURE (-1)                /* Function failure return val */
#define SUCCESS 0                   /* Function success return val */
#define FOREVER for (;;)            /* Infinite loop declaration   */
#define EOS     '\0'                /* End of string value         */

#ifndef NULL
#define NULL    0L                  /* Null long value             */
#endif

#ifndef EOF
#define EOF     (-1)                /* EOF value                   */
#endif

#endif /* __PORTAB__ */

/*************************************************************************/
```

Das Benutzen dieser Datei ist denkbar einfach. Lesen Sie sie dazu in einen Texteditor ein und ändern Sie 3 Ziffern. Dann können Sie Ihr Programm in jeder GEM-Umgebung übersetzen und laufen lassen.

Zuerst ändern Sie das Betriebssystem ab. Falls die „1" mit Ihrem Betriebssystem nicht übereinstimmt, ersetzen Sie sie durch eine „0" und die entsprechende „0" durch eine „1". Dasselbe machen Sie mit dem Prozessor sowie dem verwendeten Compiler.

Falls Sie nicht die Default-GEM-Version des jeweiligen Betriebssystems benutzen, ändern Sie die GEM-Definition entsprechend ab. Jetzt können Sie schon Ihr erstes GEM-Programm übersetzen lassen.

Wenn Sie Include-Dateien des Compilers benutzen, sollten diese zuerst eingebunden werden. Danach können PORTAB und die GEM-Header VDI und AES benutzt werden. Beispiel:

```
#include <stdio.h>  /* Compiler-spezifisch */
#include <string.h> /* Compiler-spezifisch */
#include <portab.h>
#include <vdi.h>
#include <aes.h>

GLOBAL WORD main ()
{
... GEM-Programm...
} /* main */
```

Wie Sie an den spitzen Klammern sehen, werden alle Dateien zunächst aus dem System-Verzeichnis des Compilers geholt. Dort hinein sollten Sie auch alle Include-Dateien aus dem Verzeichnis MISC auf der Beispieldiskette kopieren, nachdem Sie die Änderungen in PORTAB.H gemacht haben.

In den folgenden Kapiteln werden wir auf die Entwicklungsumgebungen der einzelnen Compiler näher eingehen. Es wird gezeigt, wie sich eine GEM-Applikation übersetzen und linken läßt. Als Beispiel wählen wir die GEM-Applikation SCRAP, die ausführlich in Kapitel 6 erläutert wird.

Für die Applikation SCRAP befinden sich alle Informationen zum Compilieren und Linken in den entsprechenden Ordnern der Betriebssysteme (GEMDOS, FLEXOS, MSDOS). In Kapitel 6.4 wird die Einrichtung der Entwicklungsumgebung für diese Applikation noch genauer erläutert.

## 3.1 ATARI GEMDOS

Das Betriebssystem von ATARI (192 KByte ROM) stammt zum größten Teil von Digital Research. Dazu gehört die grafische Benutzeroberfläche GEM und das Disk-Operating-System GEMDOS. Die BIOS- und XBIOS-Funktionen von ATARI machen nur einen Bruchteil der 192 KByte aus. Wenn von GEMDOS die Rede ist, meinen wir also immer die GEM-Version 1.X in den ROMs eines ATARI ST.

Darüber hinaus kann man auch die GEM-Version 2.2 von ABC-Software erwerben, die dann auf dem GEMDOS aufsetzt. Sie müßten in der Datei PORTAB.H lediglich die Zeile „#define GEM GEM1“ ersetzen durch:

```
#define GEM GEM2
```

Auf dem ATARI ST können Accessories auch als Programm gestartet und per Variable (_app) abgefragt werden. Im Startup-Code eines Compilers muß dies allerdings unterstützt werden. Wo das nicht der Fall ist, werden die Quellen der angepaßten Startup-Programme mitgeliefert.

Um die verschiedenen GEM-Versionen besser zusammenfassen zu können, wurden zwei neue Header-Dateien erstellt. Die Dateien VDI.H und AES.H beinhalten alle GEM 1.X, GEM 2.X, GEM/3 und X/GEM Funktionen und Strukturen, die benötigt werden. Für ANSI-Compiler sind Prototypen angegeben. Sie ersetzen die alten Dateien OBDEFS.H und GEMDEFS.H. Sie müssen jeweils auf die Entwicklungsumgebung des Compilers gespielt werden. Dies wird im einzelnen bei den jeweiligen Compilern weiter unten angegeben.

### 3.1.1 Digital Research C

Dies war der erste Compiler, der für den ATARI ST existierte. Er wurde beim sogenannten „Entwicklungspaket" 1985 ausgeliefert und wurde dort mit Namen CCC (Compatible C-Compiler) beschrieben. Alle Programmierer, die mit diesem Compiler arbeiten mußten, können Ihnen ein Klagelied darüber singen. Wir haben selbst die Erfahrung machen müssen. Als wir noch bei der ADI Software GmbH arbeiteten, erlebten wir beim Anpassen des ADIMENS-Kerns und beim Erstellen von ADIMENS selbst die bösesten Überaschungen. Wir mußten teilweise den Assembler-Quelltext, den der Compiler erzeugt hatte, nach Fehlern untersuchen.

Die Zeiten sind aber vorbei. Inzwischen gibt es von ATARI den Nachfolger, der sich mit Alcyon meldet, aber sonst eine neuere Version des Digital Research Compiler ist.

Der Compiler besteht aus 3 Pässen, die Assembler-Quelltext erzeugen, einem Assembler (AS68) und einem Linker (Link68 von Digital Research). Um Programme zu übersetzen, sollte man sich also eine Batchdatei schreiben und diese aus einem Kommando-Interpreter aus aufrufen. Solch eine Batchdatei lautet:

```
cp68 %1.c %1.i
c068 %1.i %1.1 %1.2 %1.3 -f
del %1.i
c168 %1.1 %1.2 %1.s
del %1.1
del %1.2
as68 -l -u %1.s
del %1.s
link68 %1.68k=gemstart,%1,vdibind,aesbind,osbind,gemlib,libf
del %1.o
relmod %1.68k
del %1.68k
```

Im Ordner \GEMDOS\DR_C befindet sich auch das modifizierte GEMSTART.S, mit dem man Programme als Accessories laufen lassen kann.
Diese Assemblerdatei sollte zuerst übersetzt werden:

```
as68 -I -u gemstart.s
```

Die resultierende Objektdatei GEMSTART.O sollte dann immer mitgelinkt werden. Dasselbe gilt für die Datei VQ_GDOS.S, die einer Applikation die Möglichkeit gibt abzufragen, ob das GDOS installiert ist oder nicht. Setzen Sie die resultierende Datei VQ_GDOS.O mit Hilfe des Archivers AR68 in die VDI-Library VDIBIND oder linken Sie die Objektdatei immer mit dazu. In letzterem Fall muß die Link-Zeile dann so aussehen:

```
link68 %1.68k=gemstart,%1,vdibind,aesbind,vq_gdos,osbind,gemlib,libf
```

Eine Abfrage mittels VQ_GDOS befindet sich im Beispielprogramm SHOWGEM.

Eigene Libraries können erstellt oder bestehende Libraries geändert werden, wenn Sie das Programm AR68.PRG benutzen. Die nachfolgende Batchdatei erstellt die Library SCRAPLIB durch Hinzufügen eines neuen Moduls und löscht anschließend die Objektdatei:

```
ar68 rv scraplib %1.o
del %1.o
```

Um das Beispielprogramm SCRAP komplett zu übersetzen, muß nur die Batchdatei c_all aufgerufen werden. Sie hat folgendes Aussehen:

```
c gemain
c initerm
c graf
c power
c edit
c image
c meta
c clipbrd
c printer
c disk
c trash
c desktop
c event
c menu
c resource
c rcm
c windows
c global
l
```

Zuerst werden alle Module compiliert und anschließend mit „l.bat" gelinkt. Die Batchdatei „l.bat" sieht wie folgt aus:

```
link68 scrap.68k=gemstart,scraplib,vdibind,aesbind,osbind,gemlib,libf
relmod scrap.68k
del scrap.68k
```

Die aufgeführte Library „scraplib" wird durch die vorangegangenen Aufrufe von „c.bat" erstellt. Zum Schluß ist das fertige Programm SCRAP.PRG zum Start verfügbar. Soll es als Accessory laufen, muß der Name in SCRAP.ACC umbenannt werden.

### 3.1.2 Laser C

Die Entwicklungsumgebung für Laser C ist wesentlich einfacher als die von Digital Research C. Dort gibt es das mächtige Kommando „make", mit dem sich automatisch große Projekte verwalten lassen.

Eine Makedatei definiert die Abhängigkeiten von Modulen untereinander. Über Makros können außerdem Namen zusammengefaßt werden. Jedes Modul MODUL.C ist z.B. abhängig von seiner Header-Datei MODUL.H. Inkludiert ein Modul auch andere Module, so ist es von diesen ebenfalls abhängig. Ändert sich das Datum beim Modifizieren eines Moduls, so werden bei einem Make-Vorgang automatisch alle Dateien neu übersetzt, die von dem modifizierten Modul abhängig sind und deren übersetzte Objekt-Datei (mit Suffix .o) ein jüngeres Datum aufweisen.

Die Makedatei der Applikation SCRAP für Laser C sieht dann wie folgt aus:

```
##### MAKEFILE für SCRAP #####

#############################################################################

NAME   = scrap

#############################################################################

CFLAGS =

LFLAGS =

APP    = .prg

#############################################################################

H      = import.h export.h global.h

OBJS   = clipbrd.o desktop.o disk.o edit.o event.o gemain.o
         global.o graf.o image.o initerm.o menu.o meta.o power.o
         printer.o rcm.o resource.o trash.o windows.o

#############################################################################

$(NAME)$(APP): $(OBJS)
   $(CC) $(OBJS) $(LFLAGS) -o $@

#############################################################################
```

```
clipbrd.o : $(H) windows.h $(NAME).h resource.h desktop.h edit.h
            image.h meta.h trash.h clipbrd.h
desktop.o : $(H) windows.h $(NAME).h clipbrd.h disk.h event.h menu.h
            printer.h trash.h desktop.h
disk.o    : $(H) windows.h $(NAME).h disk.h
edit.o    : $(H) windows.h $(NAME).h resource.h desktop.h clipbrd.h
            edit.h
event.o   : $(H) windows.h desktop.h menu.h event.h
gemain.o  : $(H) initerm.h event.h gemain.h
global.o  : $(H)
graf.o    : $(H) windows.h $(NAME).h graf.h
image.o   : $(H) windows.h $(NAME).h resource.h clipbrd.h image.h
initerm.o : $(H) windows.h resource.h menu.h event.h desktop.h
            trash.h disk.h printer.h edit.h clipbrd.h image.h meta.h
            graf.h power.h initerm.h
menu.o    : $(H) windows.h $(NAME).h desktop.h graf.h power.h
            resource.h menu.h
meta.o    : $(H) windows.h $(NAME).h resource.h clipbrd.h meta.h
power.o   : $(H) windows.h $(NAME).h power.h
printer.o : $(H) windows.h $(NAME).h printer.h
rcm.o     : rcm.h
resource.o: $(H) $(NAME).h resource.h
trash.o   : $(H) windows.h $(NAME).h desktop.h clipbrd.h trash.h
windows.o : $(H) windows.h

#########################################################################
```

NAME ist ein Makro und hat den Inhalt „scrap“. Der Name der Applikation muß also eingetragen werden.

CFLAG und LFLAGS beinhalten Compiler- bzw. Linkerflags, die zusätzlich gesetzt werden können. So kann z.B. bei CFLAGS noch -Z angegeben werden, wenn der Debugger benutzt werden soll.

APP muß den Suffix für den Programmnamen beinhalten. Auf einem ATARI ST wird dieser Suffix meist „.PRG“ sein. Soll das Programm nur als Accessory übersetzt werden, so wird man APP auf den Wert „.ACC“ setzen.

H beinhaltet die 3 Header-Dateien IMPORT.H, EXPORT.H und GLOBAL.H. Diese 3 Dateien werden in jedem Modul benötigt, d.h. alle Module sind von diesen 3 Dateien abhängig.

OBJS beinhaltet den Objektcode aller Module der gesamten Applikation, sind also übersetzte Dateien mit Suffix „.O“.

Jetzt folgen die sogenannten Regeln (rules). Die Regel, die hier angewendet werden soll, lautet im Klartext:

„Die Applikation, die von den Objektdateien clipbrd.o, desktop.o,... usw. abhängt, soll mit diesen Dateien neu gelinkt werden, wenn das Programm älter ist als die Objektdateien. Der Linker soll zusätzliche Flags benutzen können."

Mit $(NAME) ist der Inhalt von NAME, also „scrap" gemeint, so daß die Regel ausführlich lauten würde:

```
scrap.prg: clipbrd.o desktop.o ...
  cc clipbrd.o desktop.o ... -o scrap.prg
```

Der Wert von LFLAGS ist in diesem Beispiel leer.

Anschließend folgen die Regeln für die einzelnen Module. Der Name des Objektcodes wird durch einen Doppelpunkt von den Header-Dateien, von denen das Modul abhängig ist, getrennt. Wie man sieht, sind alle Module von den Header-Dateien „import.h", „export.h" und „global.h" abhängig. Diese Dateien wurden ja mit dem Makro H abgekürzt, womit sich durch die Verwendung von $(H) Schreibaufwand einsparen läßt.

Startet man den Laser C-Compiler, so muß man nur noch im Menü „Make" den Menüpunkt „Set make" anwählen und die oben aufgeführte Datei „Makefile" angeben. Durch Anwählen von „Make" oder „Control-M" wird das gesamte Projekt übersetzt und das ausführbare Programm erzeugt. Jede Änderung an einem Modul mit anschließendem Make übersetzt nur das geänderte Modul und linkt dann neu.

Vor dem ersten Übersetzungslauf kopieren Sie noch die beiden Dateien VDI.H und AES.H in den Ordner \MEGAMAX\HEADERS. Sie können dann für jedes GEM-Programm benutzt werden.

### 3.1.3 Lattice C

Mit dem Lattice-Compiler können die beiden Dateien VDI.H und AES.H nicht benutzt werden. Der Compiler kommt mit den Prototypen nicht zurecht, und Makros, die länger als eine Zeile sind (trotz Benutzung des Backslash „\") kann er nicht verarbeiten. Deshalb kopieren Sie die Dateien GEMDEFS.H und OBDEFS.H auf Ihre Lattice-Entwicklungsumgebung und nennen Sie sie um in VDI.H und AES.H.

Ansonsten compilieren Sie alle Module (CLIPBRD.C, DESKTOP.C, usw.) wie im Lattice-Handbuch beschrieben und linken mit folgender Kontrolldatei:

```
*
* Standard control file for linking Lattice C modules.
*
* Step 1 - initialisation
* =======================
*
* C initialisation must be included first.
*
INPUT   startup.bin
*
* Step 2 - user modules
* =====================
*
* Now include a single user module
* (from the command line).
*
* INPUT *
INPUT clipbrd.bin
INPUT desktop.bin
INPUT disk.bin
INPUT edit.bin
INPUT event.bin
INPUT gemain.bin
INPUT global.bin
INPUT graf.bin
INPUT image.bin
INPUT initerm.bin
INPUT menu.bin
INPUT meta.bin
INPUT power.bin
INPUT printer.bin
INPUT rcm.bin
INPUT resource.bin
INPUT trash.bin
INPUT windows.bin
*
* For each extra module you want to include in the
* link include a line of the form:
*
* INPUT <file name>
*
* Step 3 - floating point library
* ===============================
*
* Comment out this line if your program does not use
* floating point
```

```
*
LIBRARY fplib.bin
*
* Step 4 - C library
* ==================
*
* C library - must always be included.
*
LIBRARY clib.bin
*
* Step 5 - GEM graphics library
* =============================
*
* GEM graphics library - only include if your program
* is trying to access graphics routines
* (by uncommenting the line).
*
LIBRARY gemlib.bin
*
```

Zu beachten wäre noch, daß die Abfrage, ob das Programm als Accessory gestartet wurde, dort nicht funktioniert, da im Modul „startup.bin" die Variable „__app" nicht gesetzt wird. Es soll aber im Public-Domain-Bereich eine neue Startup-Datei existieren.

### 3.1.4 Mark Williams C

Zunächst kopieren Sie aus dem Ordner MISC die Dateien VDI.H und AES.H in den Ordner INCLUDE ihrer Mark-Williams-Entwicklungsumgebung. Aus dem Ordner \GEMDOS\MW_C kopieren Sie die Datei STRING.H ebenfalls in den Ordner INCLUDE.

Da auch beim Mark-Williams-Compiler keine Applikationen von Accessories unterschieden werden können, entschieden wir uns, die Startup-Datei CRTS0.S, die als Quelltext im Compilerpaket mitgeliefert wird, zu modifizieren. Dort wird die Variable „__app" richtig gesetzt.

Kopieren Sie also die Datei CRTS0.S auf ihren Mark-Williams-Ordner in den Ordner LIB und lassen diese Datei mit Hilfe des Befehls

```
cc -c crts0.s
```

neu übersetzen. Die resultierende Objektdatei CRTS0.O muß sich dann im LIB-Ordner befinden.

Kopieren Sie auch die Datei STRING.H in den INCLUDE-Ordner Ihrer Entwicklungsumgebung. Zum Schluß müssen noch die beiden fehlenden Objektmodule FMBUT-

TON.O sowie FMKEYBD.O in die Library LIBAES.A kopiert werden. Dazu bedienen wir uns des Programmes AR. Zuerst wechseln wir in den Ordner LIB und rufen auf:

```
cd lib
ar rsv libaes.a fmbutn.o fmkeybd.o
cd ..
```

Zum Übersetzen der SCRAP-Applikation kann dann die Makedatei „makefile" benutzt werden. Sie ist identisch mit der Laser-C-Version. Lediglich die Compilerflags (CFLAGS) sowie die Linkerflags (LFLAGS) werden belegt.

```
CFLAGS = -VPEEP
LFLAGS = -x -s -laes -lvdi
```

### 3.1.5 Turbo C

Die Installation ist hier recht unproblematisch. Kopieren Sie die Dateien PORTAB.H, VDI.H sowie AES.H auf den Ordner \TC\INCLUDE Ihrer Festplatte oder Arbeitsdiskette. Wir haben hier angenommen, daß sich die Turbo-C-Entwicklungsumgebung im Ordner \TC befindet. Dieser wird auf MS-DOS-Rechnern standardmäßig benutzt.

Das einzige Problem besteht beim direkten Aufruf von VDI-Befehlen über die GEM-Funktion vdi(). Die Entwickler von Turbo C auf dem ATARI ST haben sich hier nicht an die Standards von Digital Research gehalten, was mehr als ärgerlich ist. Sie ignorierten die Original-Bindings einfach.

Füllt man bei VDI-Befehlen die Felder contrl, intin und ptsin selbst, so kann man normalerweise vdi() aufrufen. Der Grafikbefehl wird dann ausgeführt. In dieser Turbo-C-Version wurde ins AES.H folgende Struktur aufgenommen:

```
#if GEMDOS
#if TURBO_C
typedef struct
{
  WORD contrl [11];
  WORD global [80];
  WORD intin [128];
  WORD ptsin [128];
  WORD intout [45];
  WORD ptsout [128];
  VOID FAR *addrin [128];
  VOID FAR *addrout [128];
} GEMPARBLK;
```

```
extern GEMPARBLK _GemParBlk;
#endif /* TURBO_C */
#endif /* GEMDOS */
```

Alle VDI- und AES-Aufrufe werden über die Variable „_GemParBlk" getätigt. Selbst wenn man den _GemParBlk füllt und danach _VdiCtrl() aufruft (statt vdi) passiert nichts. Die VDI-Schnittstelle von Turbo C ist also total unbrauchbar. Aus diesem Grunde und vor allem um portabel für alle anderen Compiler zu sein, haben wir die von Digital Research definierte Standard-Schnittstelle einfach selbst geschrieben. Es ist folgende kleine Assemblerdatei, die sich unter dem Namen VDICALL.S auf der Diskette befindet:

```
*       VDICALL.S
*
*       for vdicalls


*>>>>>> Export references <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

        .EXPORT vdi

*>>>>>>Import references<<<<<< <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

        .IMPORT contrl
        .IMPORT intin
        .IMPORT ptsin
        .IMPORT intout
        .IMPORT ptsout

*>>>>>>Data segment<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

        .BSS

*>>>>>>>Initialized data segment <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

        .DATA

pblock: .DC.L contrl
        .DC.L intin
        .DC.L ptsin
        .DC.L intout
        .DC.L ptsout

*>>>>>>> Code segment <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

        .CODE

******** VDI ************************************************************

vdi:    MOVE.L #pblock,D1
        MOVE.W #$73,D0
```

```
        TRAP    #2
        RTS

******** Module end ************************************************

        .END
```

Wie Sie sehen, wurde hier die VDI-Schnittstelle unter Einhaltung der Konventionen aus Kapitel 2 realisiert (Adresse des Parameterblockes in Register D1, Wert 115=$73 in Register D0). Die resultierende Objektdatei VDICALL.O befindet sich für alle diejenigen, die den Turbo-C-Assembler nicht besitzen, ebenfalls auf Diskette. Wenn Sie direkte VDI-Aufrufe tätigen möchten, wie dies in VDITEST, SHOWGEM und SCRAP getan wird, so linken Sie über die entsprechende Projektdatei das Modul VDICALL.O immer mit dazu. Sie können VDICALL.O auch in Ihre GEM-Library TCGEMLIB.LIB mit aufnehmen. Dann ersparen Sie sich die Angabe in der Projektdatei.

Die Projektdatei für die Applikation SCRAP sieht (ohne Abhängigkeiten) wie folgt aus:

```
                ; SCRAP.PRJ
                ; ----------
.C[-W-par -W-sig]
=               ; list of modules follows...
tcstart.o       ; startup code
vdicall.o       ; direct vdi calls
clipbrd
desktop
disk
edit
event
gemain
global
graf
image
initerm
menu
meta
power
printer
rcm
resource          [-W-rpt]
trash
windows
tcfltlib.lib    ; floating point lib
tcstdlib.lib    ; standard lib
tcextlib.lib    ; extended lib
tctoslib.lib    ; TOS lib
tcgemlib.lib    ; AES and VDI lib
```

Für alle Module mit Suffix „.C“ werden folgende Warnmeldungen abgeschaltet:

-W-par = Parameter werden in Funktion nicht benutzt
-W-sig = Verlust von signifikanten Stellen beim impliziten Cast-Operator.

Die erste Warnmeldung wird ausgeschaltet, damit gewisse Standardfunktionen in SCRAP immer die gleiche Schnittstelle haben, obwohl vielleicht der eine oder andere Parameter nicht benötigt wird.

Die zweite Warnmeldung tritt z.B auf, wenn ein float-Wert einem int-Wert zugewiesen wird. Beispiel:

```
FLOAT f;
LONG  l;
WORD  i;

f = 3.12;
l = 12;
i = f; /* hier Warnung von Verlust signifikanter Stellen */
i = l; /* auch hier Warnung */
```

Vor allem in der Applikation SHOWGEM werden häufig bewußt solche impliziten Typumwandlungen vorgenommen. Die Cast-Operation

```
i = (WORD)f;
i = (WORD)l;
```

hätte die Warnmeldungen verhindert, was uns aber zuviel Schreibaufwand war.

## 3.2 MS-DOS und FlexOS

MS-DOS und FlexOS laufen auf Rechnern der INTEL-Familie. Dort muß lediglich beachtet werden, daß die GEM-Programme auf den Disketten im sogenannten Large-Modell übersetzt werden.

Bei Benutzung von Bit-Image-Dateien muß der Kopf nach dem Einlesen wort- bzw. langwortweise vertauscht werden (siehe SHOWGEM und Kapitel 2.5), da er im 68000-Modell vorliegt.

### 3.2.1 Microsoft C

Kopieren Sie zunächst die Dateien PORTAB.H, VDI.H und AES.H in den Ordner INCLUDE, in dem sich Ihr Microsoft C-Compiler befindet. Vergessen Sie aber nicht, in der Datei PORTAB.H Ihr Betriebssystem sowie Ihren Compiler und den Prozessortyp
14 (I8086) einzustellen.

Zum Übersetzen der SCRAP-Applikation kann dann die Makedatei „makefile“ benutzt werden.

```
##### MAKEFILE für SCRAP #####

#############################################################################

NAME   = scrap

#############################################################################

CFLAGS = -AL -Oas -Gs

LFLAGS = /ST:12288 /SE:512 /NOI /E

APP    = .app

#############################################################################

.c.obj:
    cl -c $(CFLAGS) $*.c

.asm.obj:
    masm /Mx $*;

#############################################################################

H      = $*.c $*.h import.h export.h global.h

OBJS   = clipbrd.obj desktop.obj disk.obj edit.obj event.obj\
         gemain.obj global.obj graf.obj image.obj initerm.obj\
         menu.obj meta.obj power.obj printer.obj rcm.obj\
         resource.obj trash.obj windows.obj

#############################################################################

clipbrd.obj : $(H) windows.h $(NAME).h desktop.h edit.h image.h\
              meta.h trash.h

desktop.obj : $(H) windows.h $(NAME).h disk.h event.h menu.h\
              printer.h trash.h clipbrd.h

disk.obj    : $(H) windows.h $(NAME).h
```

```
edit.obj     : $(H) windows.h $(NAME).h resource.h desktop.h\
               clipbrd.h

event.obj    : $(H) windows.h desktop.h menu.h

gemain.obj   : $(H) initerm.h event.h

global.obj   : $(H)

graf.obj     : $(H) windows.h $(NAME).h

image.obj    : $(H) windows.h $(NAME).h resource.h clipbrd.h

initerm.obj  : $(H) windows.h $(NAME).h resource.h menu.h event.h\
               desktop.h trash.h disk.h printer.h clipbrd.h edit.h\
               image.h meta.h graf.h power.h

menu.obj     : $(H) windows.h $(NAME).h desktop.h graf.h power.h\
               resource.h

meta.obj     : $(H) windows.h $(NAME).h resource.h clipbrd.h

power.obj    : $(H) windows.h $(NAME).h resource.h

printer.obj  : $(H) windows.h $(NAME).h

rcm.obj      : $(H)

resource.obj: $(H) $(NAME).h

trash.obj    : $(H) windows.h $(NAME).h desktop.h clipbrd.h

windows.obj  : $(H)

#########################################################################

$(NAME)$(APP) : $(OBJS)
    link $(LFLAGS) @$(NAME).lnk

#########################################################################
```

Die Makedatei sieht fast so aus wie für Laser C oder Mark Williams C. Hier wird von den Compiler- und Linkerflags Gebrauch gemacht. Beachten Sie die Übersetzung im Large-Modell (-AL) sowie die möglichen Optimierungen (-Oas -Gs).

Für die Applikation wird beim Linken ein Stack von 12 KByte reserviert. Das Suffix unter MS-DOS lautet immer „.APP".

Anschließend folgt die Regel für die Übersetzung von C-Dateien in Objektdateien. Es wird der Compiler (cl) aufgerufen, der ein Modul nur übersetzen soll (-c = compile only). Für Assemblermodule (Dateien mit Suffix „.ASM") gibt es eine entsprechende Regel.

Schließlich folgen die Makros für die Header- und Objektdateien wie in den anderen Makedateien weiter oben.

### 3.2.2 Turbo C

Kopieren Sie zunächst die Dateien PORTAB.H, VDI.H und AES.H in den Ordner INCLUDE, in dem sich Ihr Turbo-C-Compiler befindet (meist der Ordner \TC). Vergessen Sie aber nicht, in der Datei PORTAB.H Ihr Betriebssystem sowie Ihren Compiler und den Prozessortyp (I8086) einzustellen.

Die Konfigurationsdateien für Turbo C werden ebenfalls mitgeliefert. Kopieren Sie sie in Ihren Ordner, in dem die Applikation entwickelt werden soll. Das Modell müßte beim Starten von Turbo C dann auf LARGE stehen.

Die Projektdatei für das Übersetzen der Applikation SCRAP (ohne Abhängigkeiten) lautet:

```
clipbrd
desktop
disk
edit
event
gemain
global
graf
image
initerm
menu
meta
power
printer
rcm
resource
trash
windows
ltcgem.lib
```

Die Library „ltcgem.lib“ muß selbst hergestellt werden. Diese Arbeit wird aber vom neuen Programmer's Toolkit von Digital Research sehr vereinfacht. Die Library enthält die VDI- und AES-Funktionen, sowie eine direkte Schnittstelle zum DOS und weitere Funktionen.

### 3.2.3 High C

Kopieren Sie zunächst die Dateien PORTAB.H, VDI.H und AES.H in den Ordner INCLUDE, in dem sich Ihr High C-Compiler befindet. Vergessen Sie aber nicht, in der Datei PORTAB.H Ihr Betriebssystem sowie Ihren Compiler und den Prozessortyp (I8086) einzustellen.

Die Makedatei für den High-C-Compiler sieht wie folgt aus:

```
#### MAKEFILE für SCRAP ####

##########################################################################

NAME   = scrap

##########################################################################

CFLAGS = -Off Prototype_overwrite_warnings -On PCC_msgs

LFLAGS =

APP    = .286

##########################################################################

COMPILE = hc $*.c $(CFLAGS)

##########################################################################

H      = import.h export.h global.h

OBJS   = clipbrd.obj desktop.obj disk.obj edit.obj event.obj\
         gemain.obj global.obj graf.obj image.obj initerm.obj\
         menu.obj meta.obj power.obj printer.obj rcm.obj\
         resource.obj trash.obj windows.obj
```

```
#############################################################################

$(NAME)$(APP): $(OBJS)
    link86 $(LFLAGS) $(NAME).inp[i]

#############################################################################


clipbrd.obj : clipbrd.c clipbrd.h windows.h $(NAME).h resource.h\
              desktop.h edit.h image.h meta.h trash.h
    $(COMPILE)

desktop.obj : desktop.c desktop.h windows.h $(NAME).h disk.h event.h\
              menu.h printer.h trash.h clipbrd.h
    $(COMPILE)

disk.obj    : disk.c disk.h windows.h $(NAME).h
    $(COMPILE)

edit.obj    : edit.c edit.h windows.h $(NAME).h resource.h desktop.h\
              clipbrd.h
    $(COMPILE)

event.obj   : event.c event.h windows.h desktop.h menu.h
    $(COMPILE)

gemain.obj  : gemain.c gemain.h initerm.h event.h
    $(COMPILE)

global.obj  : global.c global.h
    $(COMPILE)

graf.obj    : graf.c graf.h windows.h $(NAME).h
    $(COMPILE)

image.obj   : image.c image.h windows.h $(NAME).h resource.h\
              clipbrd.h
    $(COMPILE)

initerm.obj : initerm.c initerm.h windows.h $(NAME).h resource.h\
              menu.h event.h desktop.h trash.h disk.h printer.h\
              clipbrd.h edit.h image.h meta.h graf.h power.h
    $(COMPILE)
```

```
menu.obj     : menu.c menu.h windows.h $(NAME).h desktop.h graf.h\
               power.h resource.h
    $(COMPILE)


meta.obj     : meta.c meta.h windows.h $(NAME).h resource.h clipbrd.h
    $(COMPILE)


power.obj    : power.c power.h windows.h $(NAME).h resource.h
    $(COMPILE)


printer.obj  : printer.c printer.h windows.h $(NAME).h
    $(COMPILE)


rcm.obj      : rcm.c rcm.h
    $(COMPILE)


resource.obj: resource.c resource.h $(NAME).h
    $(COMPILE)


trash.obj    : trash.c trash.h windows.h $(NAME).h desktop.h clipbrd.h
    $(COMPILE)


windows.obj  : windows.c windows.h
    $(COMPILE)


##############################################################################
```

# 4 Aufgabenstellung eines kleinen GEM-Programmes

In diesem Kapitel wird ein GEM-Programm vorgestellt, das als Ausgabeprogramm universell einsetzbar ist. Gezeigt werden soll vor allem das Handling von Dialogboxen, Fehlermeldungen und das Benutzen der MS-DOS- bzw. GEMDOS-Funktionen zur Erhöhung der Portabilität.

Außerdem werden benutzerdefinierte Objekte in einer Dialogbox verwendet, um die Technik klarzumachen. Auch die Transformation von Piktogrammen und Bildern vom Standardformat ins gerätespezifische Format wird gezeigt.

Die Hauptaufgabe des Programmes ist die Ausgabe von beliebigen Vektor- und Pixelgrafiken, die im GEM-Format (Kapitel 2.5) vorliegen, auf eines der folgenden Ausgabegeräte:

- Bildschirm
- Plotter
- Drucker
- Kamera

Werden Vektorgrafiken ausgegeben, so soll eine exakte Skalierung stattfinden. Eine Linie von 5 Zoll Länge soll auch genau so auf einem Drucker, Plotter oder auf einer Kamera wiedergegeben werden.

Für den Bildschirm soll ebenfalls die Möglichkeit der exakten Skalierung bestehen. Dies ist aber nicht ohne weiteres möglich. Beim Öffnen einer Arbeitsstation erhält man zwar die Anzahl der Pixel in x- und y-Richtung, sowie die Breite und Höhe eines einzelnen Pixel in 1/1000 mm. Die Werte gelten aber nicht für einen Bildschirm. Das VDI weiß nicht, ob z.B. ein Bildschirm von IBM oder ein NEC Multisync angeschlossen ist. Daher läßt sich über die exakte Größe eines Bildschirmpunktes nur wenig aussagen. Lediglich der sogenannte „Aspect ratio", also das Verhältnis von x zu y läßt sich bestimmen.

Es bleibt nur ein Lösungsweg: Ausmessen der Größe des sichtbaren Bildschirmes. Das Programm kann dann aus der Anzahl der Pixel die wirkliche Größe eines Bildpunktes bestimmen.

Bei der Ausgabe von Pixelgrafiken sollen nicht nur monochrome Bilder korrekt angezeigt werden. Jede Kombination soll möglich sein. Eine Grafik mit 16 Farben auf einem Bildschirm mit 4 Farben, eine monochrome Grafik auf einem Farbbildschirm usw. Dabei sollen Farbbilder, die n Farben beinhalten, auf einen Bildschirm mit weniger als n Farben konvertiert werden.

Zur Darstellung einer Grafik (Vektor oder Pixel) soll eine Datei mit Suffix „.GEM" über die GEM-Dateiauswahlbox angewählt werden. Pixelgrafiken mit Suffix „.IMG" werden auch über die gleichnamigen Dateien mit Suffix „.GEM" angewählt. Zu jeder Pixelgrafik erzeugt GEMPAINT nämlich auch eine Datei mit Suffix „.GEM", die Zusatzinformationen, wie die tatsächliche Ausdehnung der Grafik in Pixel, enthält.

## 4.1 Die Bedienung von SHOWGEM

Nach dem Start des Programmes erscheint die GEM-Dateiauswahlbox. Es werden alle Dateien aus dem aktuellen Ordner, also aus dem Ordner, aus dem SHOWGEM gestartet wurde, angezeigt.

Dort kann man eine beliebige GEM-Datei anwählen und den OK-Knopf drücken. Anschließend erscheint eine Dialogbox, die die Einstellung von Ausgabeparametern erlaubt.

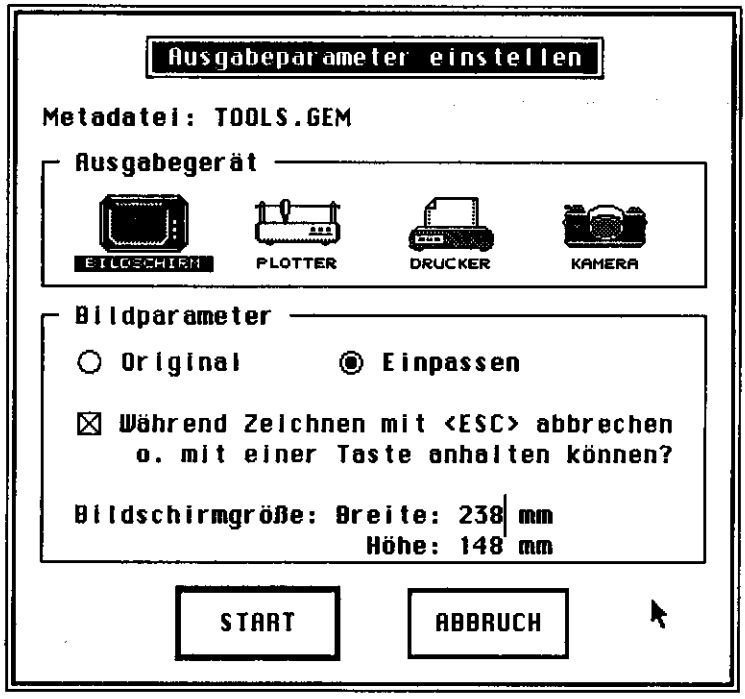



**Abb. 4.1:** Dialog von SHOWGEM

Man kann jetzt sofort den Knopf START drücken, um sich die Grafik auf dem Bildschirm anzeigen zu lassen. Man kann aber auch direkt mit der Maus Änderungen vornehmen.

Wählen Sie zunächst das gewünschte Ausgabegerät an. Standardmäßig ist der Bildschirm selektiert. Sie können aber mit der Maus auch auf ein anderes Gerät wechseln. Wenn Sie es anklicken, erscheint das gewählte Gerät invertiert.

Anschließend können Sie Bildparameter einstellen. Sie gelten nur für Metadateien, also Vektorgrafiken, da diese auf einfache Weise skaliert werden können.

Sie können unter 2 Ausgabeanpassungen wählen, wobei Einpassen vorbelegt ist:

- Original
- Einpassen

Wird Original gewählt, so wird die Vektorgrafik in der Originalgröße ausgegeben. Das bedeutet, daß Längenbeziehungen erhalten bleiben. Eine Linie von 1 cm in der Grafik bedeutet, daß sie mit der gleichen Länge auf das Ausgabegerät gebracht wird. Hier passiert es häufig, daß die Grafik nie ganz auf den Bildschirm paßt. Wurde z.B. in GEM-DRAW eine Grafik im DIN A4-Format Hochformat erzeugt, so wird bei der Ausgabe (meist auf den Bildschirm) das Bild eventuell abgeschnitten.

Wird Einpassen gewählt, so wird die Vektorgrafik an das Ausgabegerät angepaßt. Eine große Grafik wird dann so weit verkleinert, bis sie ganz auf das Ausgabegerät paßt. Ist eine Grafik kleiner als die Fläche auf dem Ausgabegerät, so wird die Grafik entsprechend vergrößert. Die Grafik paßt sich also exakt dem Gerät an.

Zusätzlich können Sie Ein- oder Ausschalten, ob beim Zeichnen der Grafik mit der Taste <ESC> abgebrochen werden kann oder nicht. Ist die Option eingeschaltet, kann mit jeder beliebigen Taste die Ausgabe der Grafik vorübergehend angehalten werden.

Falls Sie die Grafik auf dem Bildschirm ausgeben möchten, können Sie auf Wunsch Ihren Bildschirm abmessen und die Breite sowie die Höhe in die entsprechenden Felder eintragen. Die Zahlen, die dort beim Start automatisch eingetragen worden sind, wurden über das VDI berechnet und können falsch sein. Auf einem VGA-Monitor beispielsweise stimmen die Maße nicht einmal annähernd mit der Realität überein. Wenn Sie die richtigen Größen eingetragen haben, erscheinen Grafiken auch auf dem Bildschirm exakt skaliert.

Nachdem Sie alle Einstellungen vorgenommen haben, können Sie auf den Knopf START drücken, und die Ausgabe beginnt. Wählen Sie den Knopf ABBRUCH, um zur Dateiauswahlbox zurückzukehren.

Wurde eine Grafik auf dem Bildschirm ausgegeben, so wartet das Programm auf das Drücken einer Taste oder eines Mausknopfes. Danach erscheint wieder die Dateiauswahlbox, und Sie können weitere Ausgaben vornehmen.

Das Programm wird abgebrochen, wenn Sie in der Dateiauswahlbox den Knopf ABBRUCH wählen.

Am Beispiel dieser Dialogbox sehen Sie gleich den Vorteil von benutzerdefinierten Objekten. Wir haben für sogenannte Radio-Buttons runde Knöpfe mit einem schwarzen Kern benutzt. Dies entspricht der Konvention auf einem Apple Macintosh oder der Konvention unter dem Presentation Manager bzw. unter Microsoft Windows. Der Benutzer sieht sofort, daß das Anwählen des Knopfes „Original" den Knopf „Einpassen" deselektiert und umgekehrt.

Die sogenannten „Checkboxen" sind ankreuzbare Kästchen. Selektiert man eines davon, so erscheint in ihm ein Kreuz. Jede Checkbox kann einzeln an- bzw. ausgeschaltet werden. Dies entspricht ebenfalls der Konvention auf einem Apple Macintosh oder der Konvention unter dem Presentation Manager bzw. unter Microsoft Windows. Auch hier sieht der Benutzer durch die Form eines Rechteckes sofort, daß das Anwählen einer Checkbox unabhängig von anderen Checkboxen erfolgt.

Die Verwaltung von runden Radio-Buttons und Checkboxen wird im Programm SHOWGEM sowie SCRAP ausführlich gezeigt. Nun ist es an Ihnen, den Software-Entwicklern, etwas mehr Standardisierung bei der Gestaltung von ansprechenden Dialogboxen hineinzubringen. Nehmen Sie sich jetzt ein Beispiel am Apple Macintosh und dem Presentation Manager. Der Benutzer wird es Ihnen danken.

## 4.2 Erläuterungen zum Programm SHOWGEM

Es folgt zunächst das komplette Listing der GEM-Applikation SHOWGEM. Anschließend werden zu den einzelnen Funktionen Erläuterungen gegeben, wo diese für das Verständnis notwendig sind.

```
/* Mini-Output Programm, das Metadateien und Bitimage-Dateien */
/* auf ein beliebiges Ausgabegerät sendet                     */

#include <stdio.h>
#include <string.h>
#include <portab.h>
#include <vdi.h>
#include <aes.h>

#if GEMDOS
#include <osbind.h>
#define Mavail() (LONG)Malloc (-1L)
#else
#include <gemdos.h>
#include <dosbind.h>
#endif
```

```
#include "showgem.h"

/****** DEFINES ********************************************************/

#define DEBUG        0
#define MAX_PLANES   8
#define MAX_PATTERNS 8

#define SCREEN       1
#define PLOTTER     11
#define PRINTER     21
#define METAFILE    31
#define CAMERA      41
#define TABLET      51

#define NDC          0
#define RC           2

#define V_CLRWK              3  /* Metafile VDI functions */
#define V_UPDWK              4
#define V_ESC                5
#define V_PLINE              6
#define V_PMARKER            7
#define V_GTEXT              8
#define V_FILLAREA           9
#define V_GDP               11
#define VST_HEIGHT          12
#define VST_ROTATION        13
#define VS_COLOR            14
#define VSL_TYPE            15
#define VSL_WIDTH           16
#define VSL_COLOR           17
#define VSM_TYPE            18
#define VSM_HEIGHT          19
#define VSM_COLOR           20
#define VST_FONT            21
#define VST_COLOR           22
#define VSF_INTERIOR        23
#define VSF_STYLE           24
#define VSF_COLOR           25
#define VSWR_MODE           32
#define VST_ALIGNMENT       39
#define VSF_PERIMETER      104
#define VST_EFFECTS        106
#define VST_POINT          107
#define VSL_ENDS           108
```

```
#define VSF_UDPAT           112
#define VSL_UDSTY           113
#define VR_RECFL            114
#define VS_CLIP             129

#define V_BAR                 1      /* Metafile VDI GDP functions */
#define V_ARC                 2
#define V_PIESLICE            3
#define V_CIRCLE              4
#define V_ELLIPSE             5
#define V_ELLARC              6
#define V_ELLPIE              7
#define V_RBOX                8
#define V_RFBOX               9
#define V_JUSTIFIED          10

#define V_EXIT_CUR            2      /* Metafile VDI ESC functions */
#define V_ENTER_CUR           3
#define V_FORM_ADV           20
#define V_OUTPUT_WINDOW      21
#define V_CLEAR_DISP_LIST    22
#define V_BIT_IMAGE          23
#define V_ALPHA_TEXT         25
#define V_TOPBOT          18501

#define MAX_ALERT  3                 /* Anzahl Fehler in Resource */
#define RSC_NAME   "SHOWGEM.RSC"       /* Name der Resource-Datei */
#define PATHSEP    '\\'

#define DHEADER   0x20                     /* user defined objects */
#define DCHECKBOX 0x40
#define DRBUTTON  0x60

#define ESC       27
#define abs(x)       ((x) <  0  ? -(x) : (x))  /* Absolut-Wert     */
#define max(x,y)    (((x) > (y)) ? (x) : (y))  /* Maximim-Funktion */
#define min(x,y)    (((x) < (y)) ? (x) : (y))  /* Minimum Funktion */

/****** TYPES ****************************************************/

typedef struct
{
  WORD bgc;
  WORD style;
  WORD interior;
  WORD bdc;
```

```
  WORD width;
  WORD chc;
} OBINFO;


typedef struct meta_header
{
  WORD id;
  WORD headlen;
  WORD version;
  WORD transform;
  WORD min_x;
  WORD min_y;
  WORD max_x;
  WORD max_y;
  WORD pwidth;
  WORD pheight;
  WORD ll_x;
  WORD ll_y;
  WORD ur_x;
  WORD ur_y;
  WORD bit_image;
} META_HEADER;


typedef struct img_header
{
  WORD version;
  WORD headlen;
  WORD planes;
  WORD pat_run;
  WORD pix_width;
  WORD pix_height;
  WORD sl_width;
  WORD sl_height;
} IMG_HEADER;


typedef struct rect
{
  WORD x;
  WORD y;
  WORD w;
  WORD h;
} RECT;
```

```
typedef struct lrect
{
  LONG x;
  LONG y;
  LONG w;
  LONG h;
} LRECT;

typedef struct drect
{
  DOUBLE x;
  DOUBLE y;
  DOUBLE w;
  DOUBLE h;
} DRECT;

typedef struct point
{
  WORD x;
  WORD y;
} POINT;

typedef BYTE STRING [82];

/****** VARIABLES ***************************************************/

#if GEMDOS
#if MW_C
LONG _stksize = 12288;      /* 12 KBytes Stack für Mark Williams C */
#endif
#endif

#if MSDOS
#if TURBO_C
UWORD _stklen = 12288;          /* 12 KBytes Stack für Turbo C */
#endif
#endif

#if DR_C | LASER_C | MW_C
EXTERN WORD        gl_apid;     /* Identifikation für Applikation */
#else
GLOBAL WORD        gl_apid;     /* Identifikation für Applikation */
#endif
```

```
GLOBAL WORD         contrl [12];      /* GEM Arrays für Parameter... */
GLOBAL WORD         intin [256];      /* ...von VDI Aufrufen */
GLOBAL WORD         ptsin [256];
GLOBAL WORD         intout [256];
GLOBAL WORD         ptsout [256];

LOCAL  WORD         n_pts;
LOCAL  WORD         n_int;
LOCAL  LRECT        out_device;
LOCAL  LRECT        src;
LOCAL  LRECT        dst;
LOCAL  DOUBLE       dst_factor;       /* Faktor zum Einpassen */
LOCAL  DRECT        aspect_factor;    /* Faktor für Maßstabstreue */
LOCAL  DRECT        src_pixel;
LOCAL  LRECT        dst_pixel;
LOCAL  POINT        origo;
LOCAL  WORD         work_in [103];
LOCAL  WORD         work_out [57];
LOCAL  WORD         vdi_handle;
LOCAL  WORD         out_handle;
LOCAL  WORD         phys_handle;
LOCAL  BYTE         meta_name [80];
LOCAL  ULONG        meta_len;
LOCAL  UWORD        meta_index;
LOCAL  UWORD        *meta_buffer;
LOCAL  META_HEADER  *meta_header;
LOCAL  BYTE         fs_iinpath [128];
LOCAL  BYTE         fs_iinsel [13];
LOCAL  WORD         fs_iexbutton;
LOCAL  WORD         more_fonts;
LOCAL  WORD         gl_wbox, gl_hbox, gl_wattr, gl_hattr;

LOCAL  OBJECT       *dialog;
LOCAL  OBJECT       *userimg;
LOCAL  OBJECT       *working;

LOCAL  BYTE         *alerts [MAX_ALERT]; /* Fehlermeldungen */
LOCAL  USERBLK      check_blk;           /* Macintosh check boxes */
LOCAL  USERBLK      radio_blk;           /* Macintosh radio buttons */

/****** PROTOTYPES ***************************************************/

EXTERN VOID       vdi               _((VOID));

#if MSDOS
EXTERN PARMBLK    *fardr_start      _((VOID));
```

```
EXTERN VOID       fardr_end       _((VOID));
#endif

#if MSDOS
LOCAL WORD CDECL draw_checkbox    _((VOID));
LOCAL WORD CDECL draw_rbutton     _((VOID));
#else
LOCAL WORD CDECL draw_checkbox    _((PARMBLK *pb));
LOCAL WORD CDECL draw_rbutton     _((PARMBLK *pb));
#endif

LOCAL BOOLEAN     file_exist      _((STRING filename));
LOCAL WORD        get_path        _((STRING path, WORD drive));
LOCAL VOID        set_clip        _((WORD x, WORD y, WORD w, WORD h));
LOCAL VOID        set_ptext       _((OBJECT *tree, WORD obj,
                                     STRING s));
LOCAL VOID        get_ptext       _((OBJECT *tree, WORD obj,
                                     STRING s));
LOCAL VOID        do_state        _((OBJECT *tree, WORD obj,
                                     UWORD state));
LOCAL VOID        undo_state      _((OBJECT *tree, WORD obj,
                                     UWORD state));
LOCAL VOID        vdi_fix         _((MFDB *pfd, VOID *theaddr,
                                     WORD wb, WORD h));
LOCAL VOID        vdi_trans       _((WORD *saddr, WORD swb,
                                     WORD *daddr, WORD dwb, WORD h));
LOCAL VOID        trans_gimage    _((OBJECT *tree, WORD obj));
LOCAL VOID        get_obinfo      _((LONG obspec, OBINFO *obinfo));
LOCAL VOID        fix_objs        _((OBJECT *tree));
LOCAL BOOLEAN     init_resource   _((VOID));
LOCAL WORD        hndl_dial       _((OBJECT *tree, WORD def, WORD x,
                                     WORD y, WORD w, WORD h));
LOCAL VOID        show_dial       _((OBJECT *tree));
LOCAL VOID        end_dial        _((OBJECT *tree));
LOCAL WORD        open_work       _((WORD device));
LOCAL VOID        close_work      _((WORD device));
LOCAL BOOLEAN     gdos_ok         _((VOID));
LOCAL VOID        get_dev_info    _((WORD vdi_handle));
LOCAL VOID        wait            _((VOID));
LOCAL VOID        flip_word       _((UBYTE *adr));
LOCAL BOOLEAN     read_meta       _((STRING meta_name));
LOCAL WORD        get_word        _((VOID));
LOCAL BOOLEAN     get_code        _((VOID));
LOCAL WORD        new_width       _((WORD width));
LOCAL WORD        new_height      _((WORD height));
LOCAL VOID        transform       _((BOOLEAN flip_x, BOOLEAN flip_y));
```

```
LOCAL BOOLEAN    select_file     _((BYTE *name, BYTE *suffix,
                                    BYTE *filename));
LOCAL VOID       get_header_info _((BOOLEAN best_fit));
LOCAL VOID       show_debug      _((BOOLEAN flip_x, BOOLEAN flip_y));
LOCAL VOID       show_bit_image  _((IMG_HEADER *img_header,
                                    UBYTE *img_buffer,
                                    UBYTE *raster_buf,
                                    UBYTE *raster_ptr,
                                    UBYTE *line_buf,
                                    LONG l_buflen,
                                    UBYTE **plane_ptr,
                                    WORD max_lines,
                                    WORD screen_planes,
                                    WORD fww));
LOCAL VOID       read_bit_image  _((STRING filename));
LOCAL VOID       show_page       _((VOID));
LOCAL VOID       do_command      _((WORD device, BOOLEAN flip_x,
                                    BOOLEAN flip_y));
LOCAL VOID       show_meta       _((WORD device, BOOLEAN best_fit,
                                    BOOLEAN usr_break));

/****** FUNCTIONS ***************************************************/

LOCAL BOOLEAN file_exist (filename)
STRING filename;

{
#if GEMDOS
  return (Fsfirst (filename, 0x00) == 0);
#else
  return (Fsfirst (filename, 0x00) > 0);
#endif
} /* file_exist */

/*******************************************************************/

LOCAL WORD get_path (path, drive)
STRING path;
WORD   drive;

{
  BYTE s [64], sep [2];
  WORD ret;

  path [0] = EOS;
```

```
  ret = Dgetpath (s, drive);

  if (*s)
  {

#if MSDOS | FLEXOS
    strcpy (path + 1, s);
    path [0] = PATHSEP;
#endif

#if GEMDOS
    strcpy (path, s);
#endif

  } /* if */

  sep [0] = PATHSEP;
  sep [1] = EOS;

  strcat (path, sep);
  return (ret);
} /* get_path */

/*****************************************************************/

LOCAL VOID set_clip (x, y, w, h)
WORD x, y, w, h;

{
  WORD pxy [4];

  pxy [0] = x;
  pxy [1] = y;
  pxy [2] = x + w - 1;
  pxy [3] = y + h - 1;

  vs_clip (vdi_handle, TRUE, pxy);
} /* set_clip */

/*****************************************************************/

LOCAL VOID set_ptext (tree, obj, s)
OBJECT *tree;
WORD   obj;
STRING s;
```

```
{
  TEDINFO *ptedinfo;

  ptedinfo = (TEDINFO *)tree [obj].ob_spec;
  strncpy (ptedinfo->te_ptext, s, ptedinfo->te_txtlen - 1);
  ptedinfo->te_ptext [ptedinfo->te_txtlen - 1] = EOS;
} /* set_ptext */

/*****************************************************************/

LOCAL VOID get_ptext (tree, obj, s)
OBJECT *tree;
WORD   obj;
STRING s;

{
  TEDINFO *ptedinfo;

  ptedinfo = (TEDINFO *)tree [obj].ob_spec;
  strcpy (s, ptedinfo->te_ptext);
} /* get_ptext */

/*****************************************************************/

LOCAL VOID do_state (tree, obj, state)
OBJECT *tree;
WORD   obj;
UWORD  state;

{
  tree [obj].ob_state |= state;          /* Status im Objekt setzen */
} /* do_state */

/*****************************************************************/

LOCAL VOID undo_state (tree, obj, state)
OBJECT *tree;
WORD   obj;
UWORD  state;

{
  tree [obj].ob_state &= ~ state;       /* Status im Objekt löschen */

} /* undo_state */

/*****************************************************************/
```

```
LOCAL VOID vdi_fix (pfd, theaddr, wb, h)
MFDB *pfd;
VOID *theaddr;
WORD wb, h;

{
  pfd->mp  = theaddr;
  pfd->fwp = wb << 3;
  pfd->fh  = h;
  pfd->fww = wb >> 1;
  pfd->np  = 1;
} /* vdi_fix */

/*****************************************************************/

LOCAL VOID vdi_trans (saddr, swb, daddr, dwb, h)
WORD *saddr;
WORD swb;
WORD *daddr;
WORD dwb;
WORD h;

{
  MFDB src, dst;

  vdi_fix (&src, saddr, swb, h);
  src.ff = TRUE;

  vdi_fix (&dst, daddr, dwb, h);
  dst.ff = FALSE;

  vr_trnfm (vdi_handle, &src, &dst);
} /* vdi_trans */

/*****************************************************************/

LOCAL VOID trans_gimage (tree, obj)
OBJECT *tree;
WORD   obj;

{
  ICONBLK *piconblk;
  BITBLK  *pbitblk;
  WORD    *taddr;
  WORD    wb, hl, type;
```

```
  type = tree [obj].ob_type;

  if (type == G_ICON)
  {
    piconblk = (ICONBLK *)tree [obj].ob_spec;
    taddr    = piconblk->ib_pmask;
    wb       = piconblk->ib_wicon;
    wb       = wb >> 3;
    hl       = piconblk->ib_hicon;

    vdi_trans (taddr, wb, taddr, wb, hl);

    taddr = piconblk->ib_pdata;
  } /* if */
  else
  {
    pbitblk = (BITBLK *)tree [obj].ob_spec;
    taddr   = pbitblk->bi_pdata;
    wb      = pbitblk->bi_wb;
    hl      = pbitblk->bi_hl;
  } /* else */

  vdi_trans (taddr, wb, taddr, wb, hl);
} /* trans_gimage */

/*************************************************************************/

LOCAL VOID get_obinfo (obspec, obinfo)
LONG   obspec;
OBINFO *obinfo;

{
  WORD colorwd;

  colorwd       = (WORD)obspec;
  obinfo->bgc   = colorwd & 0x0F;
  obinfo->style = (colorwd & 0x70) >> 4;

  if (obinfo->style == 0)
    obinfo->interior = 0;
  else
    if (obinfo->style == 7)
      obinfo->interior = 1;
    else
      obinfo->interior = (colorwd & 0x80) ? 3 : 2;
```

```
  obinfo->bdc   = (colorwd & 0xF000) >> 12;
  obinfo->width = (WORD)(obspec >> 16) & 0xFF;

  if (obinfo->width > 127) obinfo->width = 256 - obinfo->width;

  obinfo->chc = (colorwd & 0x0F00) >> 8;
} /* get_obinfo */

/*****************************************************************/
/* Zeichnet ankreuzbare Buttons                                  */
/*****************************************************************/

#if MSDOS
LOCAL WORD CDECL draw_checkbox ()
{
  PARMBLK *pb;
#else
LOCAL WORD CDECL draw_checkbox (pb)
PARMBLK *pb;

{
#endif

  LONG    ob_spec;
  WORD    ob_x, ob_y, ob_width, ob_height;
  BOOLEAN selected, changed;
  WORD    pxy [10];
  OBINFO  obinfo;

#if MSDOS
  pb = fardr_start();
#endif

  ob_spec   = pb->pb_parm;
  ob_x      = pb->pb_x;
  ob_y      = pb->pb_y;
  ob_width  = pb->pb_w;
  ob_height = pb->pb_h;
  selected  = pb->pb_currstate & SELECTED;
  changed   = (pb->pb_currstate ^ pb->pb_prevstate) & SELECTED;

  get_obinfo (ob_spec, &obinfo);
  set_clip (pb->pb_x, pb->pb_y, pb->pb_w, pb->pb_h);
```

```
  vsl_type (vdi_handle, FIS_SOLID);
  vsl_ends (vdi_handle, SQUARED, SQUARED);
  vsl_width (vdi_handle, 1);
  vsl_color (vdi_handle, obinfo.bdc);
  vswr_mode (vdi_handle, MD_REPLACE);

  if (! changed) /* it was an objc_draw, so draw box */
  {
    pxy [0] = ob_x;
    pxy [1] = ob_y;
    pxy [2] = ob_x + ob_width - 1;
    pxy [3] = ob_y;
    pxy [4] = pxy [2];
    pxy [5] = ob_y + ob_height - 1;
    pxy [6] = ob_x;
    pxy [7] = pxy [5];
    pxy [8] = ob_x;
    pxy [9] = ob_y;

    v_pline (vdi_handle, 5, pxy);
  } /* if */

  if (selected) /* it was an objc_change */
    vsl_color (vdi_handle, obinfo.bgc);
  else
    vsl_color (vdi_handle, WHITE);

  set_clip (ob_x + 1, ob_y + 1, ob_width - 2, ob_height - 2);

  pxy [0] = ob_x;
  pxy [1] = ob_y;
  pxy [2] = ob_x + ob_width - 1;
  pxy [3] = ob_y + ob_height - 1;
  v_pline (vdi_handle, 2, pxy);

  pxy [0] = ob_x + ob_width - 1;
  pxy [1] = ob_y;
  pxy [2] = ob_x;
  pxy [3] = ob_y + ob_height - 1;
  v_pline (vdi_handle, 2, pxy);

#if MSDOS
  fardr_end ();
#endif
```

```
  return (pb->pb_currstate & ~SELECTED);
} /* draw_checkbox */

/*****************************************************************/
/* Zeichnet runde Radiobuttons                                   */
/*****************************************************************/

#if MSDOS
LOCAL WORD CDECL draw_rbutton ()
{
  PARMBLK *pb;
#else
LOCAL WORD CDECL draw_rbutton (pb)
PARMBLK *pb;

{
#endif

  LONG    ob_spec;
  WORD    ob_x, ob_y;
  BOOLEAN selected;
  MFDB    s, d;
  BITBLK  *bitblk;
  WORD    robj; /* radio button object number */
  WORD    pxy [8];
  WORD    index [2];
  OBINFO  obinfo;

#if MSDOS
  pb = fardr_start();
#endif

  ob_spec   = pb->pb_parm;
  ob_x      = pb->pb_x;
  ob_y      = pb->pb_y;
  selected  = pb->pb_currstate & SELECTED;

  get_obinfo (ob_spec, &obinfo);
  set_clip (pb->pb_x, pb->pb_y, pb->pb_w, pb->pb_h);

  if (selected) /* it was an objc_change */
    robj = (gl_hbox > 8) ? RBHSEL : RBLSEL; /* high res:low res */
  else
    robj = (gl_hbox > 8) ? RBHNORM : RBLNORM; /* high res:low res */
```

```
  bitblk = (BITBLK *)userimg [robj].ob_spec;

  d.mp  = NULL; /* screen */
  s.mp  = (VOID *)bitblk->bi_pdata;
  s.fwp = bitblk->bi_wb << 3;
  s.fh  = bitblk->bi_hl;
  s.fww = bitblk->bi_wb >> 1;
  s.ff  = FALSE;
  s.np  = 1;

  pxy [0] = 0;
  pxy [1] = 0;
  pxy [2] = s.fwp - 1;
  pxy [3] = s.fh - 1;
  pxy [4] = ob_x;
  pxy [5] = ob_y;
  pxy [6] = ob_x + pxy [2];
  pxy [7] = ob_y + pxy [3];

  index [0] = obinfo.bgc;
  index [1] = WHITE;

  vrt_cpyfm (vdi_handle, MD_REPLACE, pxy, &s, &d, index);

#if MSDOS
  fardr_end ();
#endif

  return (pb->pb_currstate & ~SELECTED);
} /* draw_rbutton */

/*************************************************************************/

LOCAL VOID fix_objs (tree)
OBJECT *tree;

{
  WORD    obj, robj;
  OBJECT  *ob;
  ICONBLK *ib;
  BITBLK  *bi;
  WORD    w_desire, h_desire;
  WORD    x_corr, y_corr, w_corr, h_corr;
  WORD    y_radio, h_radio;
  BOOLEAN hires;
  UWORD   xtype;
```

```
hires    = (gl_hbox > 8) ? TRUE : FALSE;
w_desire = 12;
h_desire = hires ? 12 : 6; /* ensures that boxes are square */

x_corr   = (2 * gl_wbox - w_desire) / 2;
y_corr   = (gl_hbox - h_desire) / 2;
w_corr   = -2 * x_corr;
h_corr   = -2 * y_corr;

robj     = hires ? RBHNORM : RBLNORM;
bi       = (BITBLK *)userimg [robj].ob_spec;
y_radio = hires ? y_corr : 0;
h_radio = bi->bi_hl;

check_blk.ub_code = draw_checkbox;
radio_blk.ub_code = draw_rbutton;

if (tree != NULL)
{
  obj = NIL;

  do
  {
    ob = &tree [++obj];

    if (ob->ob_type == G_ICON)
    {
      ib = (ICONBLK *)ob->ob_spec;                /* Objekthöhe = */
      ob->ob_height = ib->ib_ytext + ib->ib_htext;   /* Iconhöhe */
      trans_gimage (tree, obj);  /* Icons an Bildschirm anpassen */
    } /* if */

    if (ob->ob_type == G_IMAGE)
    {
      bi = (BITBLK *)ob->ob_spec;
      ob->ob_height = bi->bi_hl;        /* Objekthöhe = Imagehöhe */
      trans_gimage (tree, obj);  /* Bitimgs an Bildsch. anpassen */
    } /* if */

    xtype = ob->ob_type >> 8;

    switch (xtype)
    {
      case DHEADER     : ob->ob_y -= gl_hbox / 2;
                         break;
```

```
        case DCHECKBOX  : check_blk.ub_parm  = ob->ob_spec;
                          ob->ob_x          += x_corr;
                          ob->ob_y          += y_corr;
                          ob->ob_width      += w_corr;
                          ob->ob_height     += h_corr;
                          ob->ob_type        = G_USERDEF;
                          ob->ob_spec        = (LONG)&check_blk;
                          break;
        case DRBUTTON   : radio_blk.ub_parm  = ob->ob_spec;
                          ob->ob_y          += y_radio;
                          ob->ob_height      = h_radio;
                          ob->ob_type        = G_USERDEF;
                          ob->ob_spec        = (LONG)&radio_blk;
                          break;
      } /* switch */
    } while (! (ob->ob_flags & LASTOB));
  } /* if */
} /* fix_objs */

/*************************************************************************/

LOCAL BOOLEAN init_resource ()

{
  WORD   i;
  STRING s;

  if (! rsrc_load (RSC_NAME))
  {
    strcpy (s, "[3][Resource-File|");
    strcat (s, RSC_NAME);
    strcat (s, "?][ EXIT ]");
    form_alert (1, s);
    return (FALSE);
  } /* if */

  if (gl_hbox > 8) /* high resolution */
    rsrc_gaddr (R_TREE, DIALOGH, &dialog);
  else
    rsrc_gaddr (R_TREE, DIALOGL, &dialog);

    rsrc_gaddr (R_TREE, USERIMG, &userimg);
    rsrc_gaddr (R_TREE, WORKING, &working);

  for (i = 0; i < MAX_ALERT; i++)
    rsrc_gaddr (R_STRING, i, &alerts [i]);
```

```
  fix_objs (dialog);
  fix_objs (userimg);
  fix_objs (working);

#if GEM & (GEM2 | GEM3 | XGEM)
  undo_state (dialog, ROOT, SHADOWED);
  do_state (dialog, ROOT, OUTLINED);
#endif

  do_state (dialog, DSCREEN, SELECTED);
  do_state (dialog, SBESTFIT, SELECTED);
  return (TRUE);
} /* init_resource */

/*************************************************************************/

LOCAL WORD hndl_dial (tree, def, x, y, w, h)
OBJECT *tree;
WORD   def;
WORD   x, y, w, h;

{
  WORD xdial, ydial, wdial, hdial;
  WORD exit_obj;

  form_center (tree, &xdial, &ydial, &wdial, &hdial);
  form_dial (FMD_START, x, y, w, h, xdial, ydial, wdial, hdial);
  form_dial (FMD_GROW, x, y, w, h, xdial, ydial, wdial, hdial);

  objc_draw (tree, ROOT, MAX_DEPTH, xdial, ydial, wdial, hdial);
  exit_obj = form_do (tree, def) & 0x7FFF;

  form_dial (FMD_SHRINK, x, y, w, h, xdial, ydial, wdial, hdial);
  form_dial (FMD_FINISH, x, y, w, h, xdial, ydial, wdial, hdial);

  undo_state (tree, exit_obj, SELECTED);
  return (exit_obj);  /* Objekt, mit dem Dialogbox verlassen wurde */
} /* hndl_dial */

/*************************************************************************/

LOCAL VOID show_dial (tree)
OBJECT *tree;
```

```
{
  WORD xdial, ydial, wdial, hdial;
  WORD x, y, w, h;

  x = y = w = h = 0;

  form_center (tree, &xdial, &ydial, &wdial, &hdial);
  form_dial (FMD_START, x, y, w, h, xdial, ydial, wdial, hdial);
  form_dial (FMD_GROW, x, y, w, h, xdial, ydial, wdial, hdial);

  objc_draw (tree, ROOT, MAX_DEPTH, xdial, ydial, wdial, hdial);
} /* show_dial */

/*****************************************************************/

LOCAL VOID end_dial (tree)
OBJECT *tree;

{
  WORD xdial, ydial, wdial, hdial;
  WORD x, y, w, h;

  x = y = w = h = 0;

  form_center (tree, &xdial, &ydial, &wdial, &hdial);
  form_dial (FMD_SHRINK, x, y, w, h, xdial, ydial, wdial, hdial);
  form_dial (FMD_FINISH, x, y, w, h, xdial, ydial, wdial, hdial);
} /* end_dial */

/*****************************************************************/

LOCAL WORD open_work (device)
WORD device;

{
  WORD i;
  WORD handle;

  for (i = 0; i < 103; i++) work_in [i] = 1;

  work_in [0]  = device; /* device handle */
  work_in [10] = RC;     /* Raster Koordinaten */
```

```
  if (device == SCREEN)
  {
    handle = phys_handle;
    v_opnvwk (work_in, &handle, work_out); /* virtuell öffnen */
  } /* if */
  else                                     /* nicht Bildschirm */
  {
    work_in [11] = OW_NOCHANGE;          /* parallel or serial port */
    v_opnwk (work_in, &handle, work_out);  /* physikalisch öffnen */
  } /* else */

  out_device.x = 0;
  out_device.y = 0;
  out_device.w = work_out [0] + 1L;
  out_device.h = work_out [1] + 1L;

  dst_pixel.w = work_out [3];
  dst_pixel.h = work_out [4];

  return (handle);
} /* open_work */

/*****************************************************************/

LOCAL VOID close_work (device)
WORD device;

{
  switch (device)
  {
    case SCREEN   : v_clsvwk (out_handle); break;
    case PLOTTER  :
    case PRINTER  : v_clswk (out_handle);  break;
  } /* switch */
} /* close_work */

/*****************************************************************/

LOCAL BOOLEAN  gdos_ok ()

{
#if GEMDOS
#if TURBO_C|LASER_C
  return (vq_gdos ());
#else
  return (TRUE);
#else
```

```
#endif /* TURBO_C */
#else
  return (TRUE);
#endif /* GEMDOS */
} /* gdos_ok */

/*****************************************************************/

LOCAL VOID get_dev_info (vdi_handle)
WORD vdi_handle;

{
  vq_extnd (vdi_handle, FALSE, work_out);

  out_device.x = 0;
  out_device.y = 0;
  out_device.w = work_out [0] + 1L;
  out_device.h = work_out [1] + 1L;

  dst_pixel.w = work_out [3];
  dst_pixel.h = work_out [4];
} /* get_dev_info */

/*****************************************************************/

LOCAL VOID wait ()

{
  WORD  msgbuf [8];
  WORD  i;
  UWORD u;

  evnt_multi (MU_KEYBD + MU_BUTTON,
              1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
              msgbuf, 0, 0, &i, &i, &i, &i, &u, &i);
} /* wait */

/*****************************************************************/

LOCAL VOID flip_word (adr)
REG UBYTE *adr;

{
  REG UBYTE c;
```

```
  c       = adr [0];
  adr [0] = adr [1];
  adr [1] = c;
} /* flip_word */

/*****************************************************************/

LOCAL BOOLEAN read_meta (meta_name)
STRING meta_name;

{
  WORD    handle;
  WORD    i;
  BOOLEAN ok;

  ok     = FALSE;
  handle = Fopen (meta_name, 0);

  if (handle >= 0)
  {
    meta_len    = Fseek (0L, handle, 2);
    meta_buffer = (UWORD *)Malloc (meta_len);
    Fseek (0L, handle, 0);

    if (meta_buffer == NULL)
      form_alert (1, alerts [NOMEMORY]);
    else
    {
      meta_len = Fread (handle, meta_len, meta_buffer);

#if M68000
      for (i = 0; i < meta_len / 2; i++)
        flip_word ((UBYTE *)&meta_buffer [i]);
#endif

      meta_header = (META_HEADER *)meta_buffer;
      meta_index  = meta_header->headlen;

      ok = TRUE;
    } /* else */

    Fclose (handle);
  } /* if */
  else
    form_alert (1, alerts [NOFILE]);
```

```
  return (ok);
} /* read_meta */

/*************************************************************************/

LOCAL WORD get_word ()

{
  return (meta_buffer [meta_index++]);
} /* get_word */

/*************************************************************************/

LOCAL BOOLEAN get_code ()

{
  WORD i;

  contrl [0] = get_word ();
  if (contrl [0] == -1) return (FALSE);

  contrl [1] = get_word ();
  contrl [3] = get_word ();
  contrl [5] = get_word ();
  contrl [6] = out_handle;

  n_pts = contrl [1] * 2;
  n_int = contrl [3];

  for (i = 0; i < n_pts; i++) ptsin [i] = get_word ();
  for (i = 0; i < n_int; i++) intin [i] = get_word ();

  return (TRUE);
} /* get_code */

/*************************************************************************/

LOCAL WORD new_width (width)
WORD width;

{
  width = width * dst_factor * aspect_factor.w;
  if (width == 0) width = 1;

  return (width);
} /* new_width */

/*************************************************************************/
```

```
LOCAL WORD new_height (height)
WORD height;

{
  height = height * dst_factor * aspect_factor.h;
  if (height == 0) height = 1;

  return (height);
} /* new_height */

/*************************************************************************/

LOCAL VOID transform (flip_x, flip_y)
BOOLEAN flip_x, flip_y;

{
  WORD    i, ix, iy, x, y;
  WORD    gdp, n;
  BOOLEAN special;

  for (i = 0; i < contrl [1]; i++)
  {
    ix = 2 * i;
    iy = ix + 1;

    if (contrl [0] == V_GDP)
    {
      gdp = contrl [5];
      n   = -1;

      switch (gdp)
      {
        case V_ARC       :
        case V_PIESLICE  : n = 3;  break;
        case V_CIRCLE    : n = 2;  break;
        case V_ELLIPSE   :
        case V_ELLARC    :
        case V_ELLPIE    :
        case V_JUSTIFIED : n = 1;  break;
        case V_BAR       :
        case V_RBOX      :
        case V_RFBOX     : n = -1; break;
      } /* switch */

      special = (i == n); /* Radius, Länge */
    } /* if */
    else
```

```
    {
      gdp = 0;
      special = FALSE;
    } /* else */

    if (special)
    {
      switch (gdp)
      {
        case V_ARC        :
        case V_PIESLICE   : /* Radius */
        case V_CIRCLE     : ptsin [ix] = ptsin [ix] * dst.w / src.w;
                            break;
        case V_ELLIPSE    :  /* Transformiere x- und y-Radius */
        case V_ELLARC     :
        case V_ELLPIE     : ptsin [ix] = ptsin [ix] * dst.w / src.w;
                            ptsin [iy] = ptsin [iy] * dst.h / src.h;
                            break;
        case V_JUSTIFIED : ptsin [ix] = ptsin [ix] * dst.w / src.w;
                            break;
      } /* switch */
    } /* if */
    else /* Bildschirmpunkte */
    {
      ptsin [ix] -= origo.x;
      ptsin [iy] -= origo.y;

      x = ptsin [ix] * dst.w / src.w;
      y = ptsin [iy] * dst.h / src.h;

      if (flip_x)
        ptsin [ix] = dst.x + dst.w - x;
      else
        ptsin [ix] = dst.x + x;

      if (flip_y)
        ptsin [iy] = dst.y + dst.h - y;
      else
        ptsin [iy] = dst.y + y;
    } /* else */
  } /* for */
} /* transform */

/*****************************************************************/
```

```
LOCAL VOID get_header_info (best_fit)
BOOLEAN best_fit;

{
  DOUBLE  x_factor, y_factor;

  if (meta_header->ll_x != 0 ||
      meta_header->ll_y != 0 ||
      meta_header->ur_x != 0 ||
      meta_header->ur_y != 0)
  {
    src.x = min (meta_header->ll_x, meta_header->ur_x);
    src.y = min (meta_header->ll_y, meta_header->ur_y);
    src.w = abs ((LONG)meta_header->ur_x -
                 (LONG)meta_header->ll_x) + 1;
    src.h = abs ((LONG)meta_header->ur_y -
                 (LONG)meta_header->ll_y) + 1;

    if (meta_header->pwidth != 0 || /* page width/height */
        meta_header->pheight != 0)
    {
      src_pixel.w = (DOUBLE)meta_header->pwidth  * 100L / src.w;
      src_pixel.h = (DOUBLE)meta_header->pheight * 100L / src.h;

      aspect_factor.w = src_pixel.w / dst_pixel.w;
      aspect_factor.h = src_pixel.h / dst_pixel.h;
    } /* if */

    meta_header->transform = (meta_header->ll_y > 0) ? RC : NDC;

    origo.x = meta_header->ll_x;

    if (meta_header->transform == NDC)
      origo.y = -meta_header->ll_y;
    else
      origo.y = meta_header->ur_y;
  } /* if */

  if (meta_header->pwidth != 0 || /* page width/height */
      meta_header->pheight != 0)
  {
    dst.x = 0;
    dst.y = 0;
    dst.w = meta_header->pwidth  * 100L / dst_pixel.w;
    dst.h = meta_header->pheight * 100L / dst_pixel.h;
```

```
    x_factor = (DOUBLE)out_device.w / dst.w;
    y_factor = (DOUBLE)out_device.h / dst.h;

    if (best_fit) dst_factor = min (x_factor, y_factor);

    dst.w = dst.w * dst_factor;
    dst.h = dst.h * dst_factor;
    dst.x = (out_device.w - dst.w) / 2; /* center picture */
    dst.y = (out_device.h - dst.h) / 2;
  } /* if */
} /* get_header_info */

/*********************************************************************/

LOCAL VOID show_debug (flip_x, flip_y)
BOOLEAN flip_x, flip_y;

{
  v_enter_cur (phys_handle);
  printf ("src: x = %5ld, y = %5ld, w = %5ld, h = %5ld\n",
  src.x, src.y, src.w, src.h);
  printf ("dst: x = %5ld, y = %5ld, w = %5ld, h = %5ld\n",
  dst.x, dst.y, dst.w, dst.h);
  printf ("out: x = %5ld, y = %5ld, w = %5ld, h = %5ld\n",
  out_device.x, out_device.y, out_device.w, out_device.h);
  printf ("origo: x = %d, y = %d\n", origo.x, origo.y);
  printf ("flip : x = %d, y = %d\n", flip_x, flip_y);
  printf ("more_fonts   = %d\n", more_fonts);
  printf ("dst_factor   = %lf\n", dst_factor);
  printf ("aspect_factor: w = %lf, h = %lf\n",
          aspect_factor.w, aspect_factor.h);
  printf ("src_pixel    : w = %lf/1000 mm, h = %lf/1000 mm\n",
          src_pixel.w, src_pixel.h);
  printf ("dst_pixel    : w = %ld/1000 mm, h = %ld/1000 mm\n",
          dst_pixel.w, dst_pixel.h);
  printf ("Mavail       : %ld\n", Mavail ());
  wait ();
  v_exit_cur (phys_handle);
  v_clrwk (phys_handle);
} /* show_debug */

/*********************************************************************/

LOCAL VOID show_bit_image (img_header, img_buffer, raster_buf,
                           raster_ptr, line_buf, l_buflen,
                           plane_ptr, max_lines, screen_planes, fww)
```

```
IMG_HEADER *img_header;
UBYTE      *img_buffer;
UBYTE      *raster_buf;
UBYTE      *raster_ptr;
UBYTE      *line_buf;
LONG       l_buflen;
UBYTE      **plane_ptr;
WORD       max_lines;
WORD       screen_planes;
WORD       fww;

{
  UBYTE      *img_ptr;
  UBYTE      *line_ptr;
  WORD       vrc; /* vertical replication count */
  WORD       bytecols; /* counter for planedata */
  UBYTE      data;    /* one byte of pixel data */
  UBYTE      pattern [MAX_PATTERNS];
  WORD       max_pattern;
  UWORD      length;
  WORD       index [2];
  WORD       idx, count;
  WORD       i, line;
  WORD       plane;
  WORD       fwb;
  MFDB       s, d;
  WORD       pxy [8];
  WORD       segments;

  s.mp  = d.mp  = (VOID *)raster_buf;
  s.fwp = d.fwp = fww * 16;
  s.fh  = d.fh  = max_lines;
  s.fww = d.fww = fww;
  s.ff  = d.ff  = TRUE; /* standard format */
  s.np  = d.np  = screen_planes;

  fwb = fww * 2;

  pxy [0] = 0;
  pxy [1] = 0;
  pxy [2] = s.fwp - 1;
  pxy [3] = s.fh - 1;
  pxy [4] = (out_device.w - pxy [2] - 1) / 2; /* center picture */
  pxy [5] = (out_device.h - pxy [3] - 1) / 2; /* center picture */
  pxy [6] = pxy [4] + pxy [2];
  pxy [7] = pxy [5] + pxy [3];
  vs_clip (out_handle, TRUE, &pxy [4]);
```

```
index [0] = BLACK;
index [1] = WHITE;

line        = 0;
max_pattern = min (img_header->pat_run, MAX_PATTERNS);
img_ptr     = img_buffer + img_header->headlen * 2;
segments    = 1 + img_header->sl_height / (max_lines + 1);

while (line < max_lines)
{
  vrc = 1;

  for (plane = 0; plane < img_header->planes; plane++)
  {
    bytecols = l_buflen;
    line_ptr = line_buf;

    while (bytecols > 0)
    {
      data = *img_ptr++;

      switch (data)
      {
        case 0    : /* vertical replication count or pattern run */
                    data = *img_ptr++;

                    if (data == 0) /* vertical replication count */
                    {
                      if (*img_ptr++ == 0xFF) vrc = *img_ptr++;
                    } /* if */
                    else /* pattern run */
                    {
                      bytecols -= data * img_header->pat_run;
                      for (i = 0; i < max_pattern; i++)
                        pattern [i] = img_ptr [i];
                      img_ptr  += img_header->pat_run;

                      while (data > 0)
                      {
                        for (i = 0; i < max_pattern; i++)
                          *line_ptr++ = pattern [i];
                        data--;
                      } /* while */
                    } /* else */
                    break;
```

```
            case 0x80 : /* bit string
                        data = *img_ptr++;
                        bytecols -= data;

                        while (data > 0)
                        {
                          *line_ptr++ = *img_ptr++;
                          data--;
                        } /* while */
                        break;
            default   : /* solid run */
                        length    = data & 0x7F;
                        bytecols -= length;
                        data      = (data & 0x80) ? 0xFF : 0;

                        while (length > 0)
                        {
                          *line_ptr++ = data;
                          length--;
                        } /* while */
                        break;
          } /* switch */
        } /* while */

#if I8086 /* flip words */
        for (i = 0; i < l_buflen / 2; i++)
          flip_word (&line_buf [i * 2]);
#endif

        idx        = plane % screen_planes;
        raster_ptr = plane_ptr [idx];

        for (count = 0; count < vrc; count++)
        {
          line_ptr = line_buf;
          for (i = 0; i < l_buflen; i++) *raster_ptr++ |= *line_ptr++;
        } /* for */
      } /* for */

      for (i = 0; i < screen_planes; i++) plane_ptr [i] += vrc * fwb;
      line += vrc;
    } /* while */

    vr_trnfm (out_handle, &d, &s);
    d.mp = NULL; /* screen */
```

```
  if (img_header->planes == 1) /* Quellbild in monochrom zeichnen */
    vrt_cpyfm (vdi_handle, MD_REPLACE, pxy, &s, &d, index);
  else /* Quellbild in Farbe zeichnen */
    vro_cpyfm (out_handle, S_ONLY, pxy, &s, &d);
} /* show_bit_image */

/*****************************************************************/

LOCAL VOID read_bit_image (filename)
STRING filename;

{
  WORD       handle;
  WORD       i, screen_planes;
  WORD       max_lines;
  LONG       planesize, l;
  IMG_HEADER *img_header;
  LONG       img_len;
  UBYTE      *img_buffer;
  UBYTE      *raster_buf;
  UBYTE      *raster_ptr;
  UBYTE      *plane_ptr [MAX_PLANES];
  LONG       rast_buflen;
  UBYTE      *line_buf;
  LONG       l_buflen;
  WORD       fww;

  handle = Fopen (filename, 0);

  if (handle >= 0)
  {
    v_clrwk (out_handle);
    img_len    = Fseek (0L, handle, 2);
    img_buffer = (UBYTE *)Malloc (img_len);
    Fseek (0L, handle, 0);

    if (img_buffer == NULL)
      form_alert (1, alerts [NOMEMORY]);
    else
    {
      img_len    = Fread (handle, img_len, img_buffer);
      img_header = (IMG_HEADER *)img_buffer;

#if I8086
      flip_word ((UBYTE *)&headlen);
      for (i = 0; i < headlen; i++) flip_word (&img_buffer [i * 2]);
#endif
```

```
      vq_extnd (out_handle, TRUE, work_out);
      screen_planes = work_out [4];
      if (img_header->planes == 1) screen_planes = 1;

      fww       = (img_header->sl_width + 15) / 16;
      l_buflen = (img_header->sl_width + 7) / 8;
      line_buf = (UBYTE *)Malloc (l_buflen);

      max_lines = img_header->sl_height;
#if MSDOS | FLEXOS
      max_lines = 65536L / (2L * fww * screen_planes);
      max_lines = min (max_lines, img_header->sl_height);
#endif
      planesize   = 2L * fww * max_lines;
      rast_buflen = planesize * screen_planes;
      raster_buf  = (UBYTE *)Malloc (rast_buflen);
      raster_ptr  = raster_buf;

      if ((line_buf == NULL) || (raster_buf == NULL))
      {
        form_alert (1, alerts [NOMEMORY]);
        return;
      } /* if */

      for (i = 0; i < screen_planes; i++)
        plane_ptr [i] = raster_buf + i * planesize;

      for (l = 0; l < rast_buflen; l++) raster_buf [l] = 0;

      show_bit_image (img_header, img_buffer, raster_buf, raster_ptr,
                      line_buf, l_buflen, plane_ptr, max_lines,
                      screen_planes, fww);
      Mfree (raster_buf);
      Mfree (line_buf);
      Mfree (img_buffer);
    } /* else */

    Fclose (handle);
  } /* if */
  else
    form_alert (1, alerts [NOFILE]);
} /* read_bit_image */

/*****************************************************************/
```

```
LOCAL VOID show_page ()

{
  WORD pxy [4];

  pxy [0] = out_device.x;
  pxy [1] = out_device.y;
  pxy [2] = out_device.x + out_device.w - 1;
  pxy [3] = out_device.y + out_device.h - 1;

  vswr_mode (out_handle, MD_REPLACE);
  vsf_interior (out_handle, FIS_SOLID);
  vsf_color (out_handle, BLACK);
  vr_recfl (out_handle, pxy); /* Bildschirm schwarz füllen */

  pxy [0] = dst.x;
  pxy [1] = dst.y;
  pxy [2] = dst.x + dst.w - 1;
  pxy [3] = dst.y + dst.h - 1;

  vswr_mode (out_handle, MD_REPLACE);
  vsf_interior (out_handle, FIS_SOLID);
  vsf_color (vdi_handle, WHITE);
  vr_recfl (out_handle, pxy); /* Ausgabegröße weiß füllen */
  vs_clip (out_handle, TRUE, pxy);
} /* show_page */

/*****************************************************************/

LOCAL VOID do_command (device, flip_x, flip_y)
WORD    device;
BOOLEAN flip_x, flip_y;

{
  WORD    i, esc;
  STRING  filename, img_name;

  esc = 0;

  if (contrl [0] == V_ESC) /* Bit-Images nicht direkt auf Bildsch. */
  {
    esc = contrl [5];
    if ((esc == V_BIT_IMAGE) && (device == SCREEN))
    {
      esc = -1;
```

```
      strcpy (filename, fs_iinpath);
      i = strlen (filename);
      while (filename [i] != PATHSEP) i--;
      filename [i + 1] = EOS;

      for (i = 0; i < contrl [3] - 5; i++)
        img_name [i] = intin [5 + i];
      img_name [i] = EOS;
      strcat (filename, img_name);

      if (file_exist (filename))
        read_bit_image (filename);
      else
        form_alert (1, alerts [NOFILE]);
    } /* if */
  } /* if */

  switch (contrl [0])
  {
    case V_ESC      :
    case V_PLINE    :
    case V_PMARKER  :
    case V_GTEXT    :
    case V_FILLAREA :
    case V_GDP      :
    case VR_RECFL   :
    case VS_CLIP    : transform (flip_x, flip_y);
                      break;
    case VSL_WIDTH  : ptsin [0] = new_width (ptsin [0]);
                      break;
    case VST_HEIGHT :
    case VSM_HEIGHT :
    case V_TOPBOT   : ptsin [1] = new_height (ptsin [1]);
                      break;
    case VST_POINT  : if (device == SCREEN) /* use vst_height */
                      {
                        vdi (); /* do vst_point */

                        contrl [0] = VST_HEIGHT;
                        contrl [1] = 1;
                        contrl [3] = 0;

                        ptsin [0] = 0;
                        ptsin [1] = dst_factor * ptsout [1];
                        if (ptsin [1] == 0) ptsin [1] = 1;
                      } /* if */
```

```
                    else
                      intin [0] = dst_factor * intin [0];
                    break;
  } /* switch */

  if (esc >= 0) vdi ();
} /* do_command */

/*****************************************************************/

LOCAL VOID show_meta (device, best_fit, usr_break)
WORD    device;
BOOLEAN best_fit;
BOOLEAN usr_break;

{
  WORD    command;
  WORD    minmax [4];
  BOOLEAN flip_x, flip_y;
  BOOLEAN ok;
  WORD    msgbuf [8];
  WORD    i, event;
  UWORD   kr;

  src_pixel.w     = dst_pixel.w;
  src_pixel.h     = dst_pixel.h;
  aspect_factor.w = 1.0;
  aspect_factor.h = 1.0;
  dst_factor      = 1.0;
  origo.x         = 0;
  origo.y         = 0;

  src.x = 0;
  src.y = 0;
  src.w = 32768L;
  src.h = 32768L;

  dst.x = out_device.x;
  dst.y = out_device.y;
  dst.w = out_device.w;
  dst.h = out_device.h;

  minmax [0] = dst.x;
  minmax [1] = dst.y;
  minmax [2] = dst.w - 1;
  minmax [3] = dst.h - 1;
```

```
  vs_clip (out_handle, TRUE, minmax);
  get_header_info (best_fit);

  flip_x = FALSE;
  flip_y = meta_header->transform != RC;

#if GEM & GEM3
    if (device != SCREEN)
    {
      if (meta_header->pwidth < meta_header->pheight)
        v_orient (out_handle, OR_PORTRAIT);
      else
        v_orient (out_handle, OR_LANDSCAPE);
    } /* else */
#endif

#if DEBUG
  show_debug (flip_x, flip_y);
#endif /* DEBUG */

  if ((device == SCREEN) &&
      (meta_header->bit_image != 1)) show_page ();

  command = 0;
  ok      = TRUE;

  while (get_code () && ok)
  {
    command++;
    do_command (device, flip_x, flip_y);

    if (usr_break)
    {
      event = evnt_multi (MU_KEYBD + MU_TIMER,
                          1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                          msgbuf, 0, 0, &i, &i, &i, &i, &kr, &i);
      if (event & MU_KEYBD)
        if ((kr & 0xFF) == ESC)
          ok = FALSE;
        else
          wait ();
    } /* if */
  } /* while */

  v_updwk (out_handle);
} /* show_meta */

/*********************************************************************/
```

```
LOCAL BOOLEAN select_file (name, suffix, filename)
BYTE *name, *suffix, *filename;

{
  WORD i;
  BYTE s [80];

  if (*suffix)                          /* Suffix ändern */
  {
    i = strlen (fs_iinpath);
    while (fs_iinpath [i] != PATHSEP) i--;
    fs_iinpath [i + 1] = EOS;
    strcat (fs_iinpath, suffix);
  } /* if */

  name [12] = EOS;
  if (*name) strcpy (fs_iinsel, name);  /* Default-Name */
  fsel_input (fs_iinpath, fs_iinsel, &fs_iexbutton);

  strcpy (s, fs_iinpath);               /* Path aufbereiten */
  i = strlen (s);
  while (s [i] != PATHSEP) i--;
  s [i + 1] = EOS;

  if (*fs_iinsel)
  {
    strcpy (filename, s);
    strcat (filename, fs_iinsel);
  } /* if */
  else
  {
    filename [0] = EOS;                 /* Keinen Dateinamen gewählt */
    fs_iexbutton = 0;                   /* Abbruch */
  } /* else */

  return (fs_iexbutton != 0);
} /* select_file */

/*****************************************************************************/

GLOBAL WORD main ()

{
  WORD    i;
  WORD    device;
  WORD    sel_device;
```

```
WORD    exit_obj;
BOOLEAN best_fit, ok;
BOOLEAN usr_break;
BYTE    filename [13];
STRING  act_path, s;
LONG    swidth, sheight;

gl_apid = appl_init ();
phys_handle = graf_handle (&gl_wbox, &gl_hbox,
                           &gl_wattr, &gl_hattr);
vdi_handle  = open_work (SCREEN);

swidth  = out_device.w * dst_pixel.w / 1000; /* in mm */
sheight = out_device.h * dst_pixel.h / 1000; /* in mm */

if (init_resource ())
{
  sprintf (s, "%3ld", swidth);
  set_ptext (dialog, SWIDTH, s);
  sprintf (s, "%3ld", sheight);
  set_ptext (dialog, SHEIGHT, s);

  get_path (act_path, 0);
  strcpy (fs_iinpath, "A:");            /* Standard-Zugriffspfad */
  sprintf (fs_iinpath, "A:%s*.GEM", act_path);
  fs_iinpath [0] += Dgetdrv ();         /* Standard-Laufwerk */
  fs_iinsel [0]   = EOS;                /* Leerer Dateiname */
  strcpy (meta_name, " ");
  strcpy (filename, " ");

  while (select_file (filename, "*.GEM", meta_name) && *meta_name)
  {
    ok = FALSE;

    if (file_exist (meta_name))
    {
      strcpy ((BYTE *)dialog [FILENAME].ob_spec, fs_iinsel);
      exit_obj = hndl_dial (dialog, 0, 0, 0, 0, 0);
      if (exit_obj == DSTART) ok = TRUE;
    } /* if */
    else
      form_alert (1, alerts [NOFILE]);

    graf_mouse (M_OFF, NULL);
```

```
if (ok && read_meta (meta_name))
{
  best_fit  = (dialog [SBESTFIT].ob_state & SELECTED) != 0;
  usr_break = (dialog [DSTOP].ob_state & SELECTED) != 0;


  sel_device = 1;
  for (i = 0; i < 4; i++)
    if (dialog [DSCREEN + i].ob_state & SELECTED)
      sel_device = i + 1;


  switch (sel_device)
  {
    case 1  : device = SCREEN;  break;
    case 2  : device = PLOTTER; break;
    case 3  : device = PRINTER; break;
    case 4  : device = CAMERA;  break;
    default : device = 0;       break;
  } /* switch */


  if (device > 0)
  {
    switch (device)
    {
      case SCREEN   : out_handle = vdi_handle;
                      get_dev_info (out_handle);


                      get_ptext (dialog, SWIDTH, s);
                      sscanf (s, "%ld", &swidth);
                      get_ptext (dialog, SHEIGHT, s);
                      sscanf (s, "%ld", &sheight);


                      dst_pixel.w = swidth * 1000 /
                                    out_device.w;
                      dst_pixel.h = sheight * 1000 /
                                    out_device.h;


                      v_clrwk (out_handle); /* Bild. löschen */
                      break;
      case PLOTTER  :
      case PRINTER  :
      case CAMERA   : out_handle = open_work (device);
                      break;
    } /* switch */
```

```
        if (out_handle == 0)
        {
          graf_mouse (M_ON, NULL);
          form_alert (1, alerts [NODEVICE]);
          graf_mouse (M_OFF, NULL);
        } /* if */
        else
        {

          if (gdos_ok ())
            more_fonts = vst_load_fonts (out_handle, 0);
          if (device != SCREEN) show_dial (working);

          show_meta (device, best_fit, usr_break);

          if (device != SCREEN) end_dial (working);
          if (gdos_ok ()) vst_unload_fonts (out_handle, 0);

          switch (device)
          {
            case SCREEN   : wait ();              break;
            case PLOTTER  :
            case PRINTER  :
            case CAMERA   : close_work (device); break;
          } /* switch */
        } /* if */
      } /* if */

      Mfree (meta_buffer);
    } /* if */

    v_clrwk (vdi_handle);
    graf_mouse (M_ON, NULL);
  } /* while */

  rsrc_free ();
} /* if */

close_work (SCREEN);
appl_exit ();
return (0);
} /* main */
```

Beginnen wir am Ende, mit dem Hauptprogramm. Dort sehen wir schon einige GEM-typische Aufruffolgen sowie das Benutzen von Dialogboxen.

– main

Zunächst wird die Applikation initialisiert, und wir besorgen uns die physikalische Kennung des Bildschirms (phys_handle). Dann wird der Bildschirm virtuell geöffnet, da wir ihn öfters benötigen. Wir bekommen das vdi_handle zurück.

Jetzt wird die Breite und Höhe des Bildschirmes in mm berechnet. Diese Werte werden später als Vorschlag in der Dialogbox erscheinen.

Die Funktion init_resource lädt die Resource-Datei und initialisiert alle Objekte. Danach werden die Breite und die Höhe in die Dialogbox gesetzt. Zunächst wird der Wert von swidth in die Zeichenkette s umgewandelt (sprintf). Anschließend wird über die Funktion set_ptext der Text s in das Feld SWIDTH (Konstante aus SHOWGEM.H, erzeugt mit dem Resource Construction Set) der Dialogbox dialog gesetzt. Die Funktion set_ptext holt sich aus dem Wert ob_spec die Adresse der Struktur vom Typ TEDINFO und setzt den Text in das Feld te_ptext. Das gleiche gilt für die Bildschirmhöhe.

Der aktuelle Pfad für die Dateiauswahlbox wird geholt und mit dem aktuellen Laufwerk verknüpft. Die while-Schleife wird so lange wiederholt, bis der Benutzer den ABBRUCH-Knopf in der Dateiauswahlbox wählt oder einen leeren GEM-Namen zurückgibt.

Existiert die Datei (file_exist), so wird der gewählte Dateiname aus fs_iinsel in die Dialogbox gesetzt. Der Dialog-Handler (hndl_dial) wird aufgerufen, und der Knopf, den der Benutzer angewählt hat, wird zurückgeliefert (exit_obj). Wenn der Benutzer den Knopf START gewählt hat, wird ok auf TRUE gesetzt. Anschließend wird die Maus versteckt.

Jetzt wird die Metadatei eingelesen, und verschiedene Werte werden aus der Dialogbox besorgt. Die Variable best_fit enthält TRUE, wenn im Status des Knopfes SBESTFIT aus dem Objektbaum dialog das Bit SELECTED gesetzt war. Entsprechendes gilt für die Variable usr_break.

In der nachfolgenden Schleife wird das Ausgabegerät bestimmt. Wenn das Bit SELECTED im Status eines der vier Piktogramme gesetzt war, wird die Variable sel_device gesetzt. Da es sich um Radio-Buttons handelt, wird garantiert, daß immer genau ein Ausgabegerät gesetzt ist.

Aus dem Wert von sel_device wird die Variable device auf die Kennung für opnwk gesetzt (SCREEN = 1, PLOTTER = 11, usw).

War die Nummer des Ausgabegeräts größer als 0 (eigentlich immer der Fall), so werden weitere Aktionen durchgeführt. Je nach Ausgabegerät werden verschiedene Initialisierungen vorgenommen. Beim Bildschirm werden aus der Dialogbox die beiden Werte für Bildschirmbreite und -höhe ausgelesen und mit ihnen die tatsächliche Pixelgröße berechnet. Anschließend wird der Bildschirm gelöscht.

Bei allen anderen Geräten wird eine physikalische Workstation (open_work) geöffnet, wobei die Kennung out_handle zurückgeliefert wird.

Ist diese Kennung 0, so konnte das Gerät nicht geöffnet werden. Die Maus wird eingeschaltet und über form_alert eine Fehlermeldung ausgegeben.

Ansonsten wird geprüft, ob das GDOS installiert ist. Die Funktion gdos_ok liefert unter MSDOS und FLEXOS immer den Wert TRUE zurück, da das GDOS dort ein fester Bestandteil des GEM ist.

Bei einem ATARI ST wird unter TURBO C die Funktion vq_gdos aufgerufen. Sie liefert TRUE, wenn das GDOS installiert ist (siehe Kap. 2.3, VDI). Bei den anderen Compilern ist diese Funktion noch nicht verfügbar. Sie wird aber als Assemblerquelltext der Diskette beigefügt. Wenn Sie sie benutzen wollen, so linken Sie das Objektmodul bitte mit. Ansonsten liefern Sie den Wert TRUE oder FALSE in gdos_ok zurück, je nachdem, ob Sie es installiert haben oder nicht.

Ist das GDOS vorhanden, so werden zusätzliche Zeichensätze geladen. Wurde die Ausgabe nicht auf den Bildschirm gesteuert, so wird eine Meldung ausgegeben, die den Ausgabevorgang signalisiert. Die Funktion show_meta sendet die Grafik korrekt skaliert an das Ausgabegerät, je nach Einstellung der Variablen best_fit.

Anschließend wird die Dialogbox vom Bildschirm entfernt, falls die Ausgabe nicht auf den Bildschirm ging. Schließlich werden die Zeichensätze wieder aus dem Speicher entfernt.

Jetzt folgen die Abschlußhandlungen. War das Ausgabegerät der Bildschirm, so wird auf einen Tastendruck oder einen Druck auf den Mausknopf gewartet. Bei allen anderen Ausgabegeräten wird die Arbeitsstation wieder geschlossen (close_work).

Schließlich wird der meta_buffer freigegeben, der in der Funktion read_meta reserviert wurde. Jetzt werden der Bildschirm gelöscht sowie die Maus wieder eingeschaltet.

Am Ende wird die geladene Resource-Datei freigegeben (rsrc_free), der Bildschirm als Arbeitsstation geschlossen (close_work), die Applikation beendet und der Wert 0 an das Programm, welches SHOWGEM aufgerufen hatte, zurückgegeben.

Nun schauen wir uns noch einige wichtige Funktionen an, die mit Metadateien und Bit-Image-Dateien zu tun haben.

- read_meta

Die Metadatei wird geöffnet und deren Länge berechnet. Wir reservieren dynamisch Speicher, um die Datei komplett einzulesen (Malloc, Fread). Läuft das Programm auf einem MC68000-Rechner (ATARI ST, ATARI TT), so wird die gesamte Metadatei wort-

weise gedreht, da sie immer im INTEL-Format vorliegt. Die Variable meta_header wird auf den Anfang des Puffers gesetzt und der meta_index auf den Beginn der VDI-Kommandos (aus headlen ersichtlich). Am Ende wird die Datei geschlossen.

– show_meta

Zunächst werden Default-Werte eingestellt und das Clipping auf die Größe des Ausgabegerätes gestellt (dst). Anschließend werden die Angaben aus dem Kopf der Metadatei ausgewertet (get_header_info). Lesen Sie bitte in dieser Funktion weiter. Dort wird das Koordinatensystem benutzt, um die Größe der Grafik zu bestimmen. Danach wird die Größe eines Pixel mit Hilfe der Werte pwidth und pheight aus der Metadatei berechnet. Mit Hilfe dieser Werte kann der Aspect-Faktor berechnet werden. Er sagt aus, um wieviel ein Bild gedehnt oder gestaucht werden muß. Liegt die Metadatei im NDC-System vor, so müssen Y-Koordinaten gespiegelt werden. Schließlich wird die Ausgabegröße des Bildes berechnet und es wird zentriert. Nun aber zurück zu show_meta. Nach dem Holen der Header-Informationen wird das Bild ab GEM/3 im Quer- oder Hochformat ausgegeben.

Wer Debug-Informationen haben möchte, muß den Wert DEBUG am Anfang des Programmes auf 1 setzen. Es lohnt sich für die Untersuchung von Metadateien in jedem Fall. Soll die Metadatei auf den Bildschirm ausgegeben werden, so wird die Seite auf dem Bildschirm weiß dargestellt (show_page).

Jetzt werden die einzelnen VDI-Codes aus dem Metadatei-Puffer geholt (get_code) und die Grafik-Befehle auf dem Bildschirm dargestellt (do_command). Falls man mit einer Taste anhalten möchte, wird die Tastatur abgefragt. Ist kein Zeichen verfügbar (MU_TIMER), so wird die Event-Schleife verlassen. Ansonsten wird auf die Taste ESC geprüft. Ist die Grafik komplett ausgegeben, so wird das Ausgabegerät aktualisiert (v_updwk), was z.B. für Drucker wichtig ist.

– do_command

In do_command wird zunächst getestet, ob es sich um einen VDI-ESC-Befehl handelt. War es der Befehl zur Ausgabe einer Bit-Image-Datei und war das Ausgabegerät der Bildschirm, so wird die Pixelgrafik über die Funktion read_bit_image ausgegeben.

Danach wird gemäß dem VDI-Befehl (contrl [0]) zu verschiedenen Aktionen verzweigt. Bei den meisten Befehlen werden jetzt die Punkt- und Radius- sowie Längenkoordinaten in die neue Ausgabegröße transformiert. Zeilenbreiten (VSL_WIDTH) sowie Texthöhen (VST_HEIGHT) werden angepaßt. Bei Punktgrößen wird auf dem Bildschirm besonders verfahren. Dort wird nicht vst_point, sondern vst_height benutzt. Damit werden Zeichen auf dem Bildschirm beliebig skaliert. Ansonsten wird die Punktgröße über den Wert dst_factor angepaßt.

– init_resource

Interessant ist noch die Funktion init_resource. Nach dem erfolgreichen Laden der Resource-Datei wird die Bildschirmauflösung getestet. Ist die Höhe eines Zeichens aus dem Standardzeichensatz größer als 8, so handelt es sich um einen hochauflösenden Bildschirm. Dann wird die Dialogbox DIALOGH, sonst die Dialogbox DIALOGL benutzt. Sie enthalten die gleichen Objekte mit Ausnahme der Piktogramme. Sie sind kleiner gestaltet, da die Dialogbox sonst nicht mehr auf den Bildschirm passen würde. Schauen Sie sich die Resource-Datei auf Diskette doch einmal genauer an.

Nachdem die Fehlermeldungen geholt wurden (R_STRING), werden alle Dialogboxen angepaßt (fix_objs). Besonders bei benutzerdefinierten Objekten müssen Aktionen durchgeführt werden. Zum Schluß wird das Piktogramm des Bildschirmes sowie der Radio-Knopf „Einpassen" vorselektiert. Ein Wert muß dort ja vorbelegt sein.

– fix_objs

Alle nötigen Anpassungen für Piktogramme sowie benutzerdefinierte Objekte werden durchgeführt. Piktogramme (Icons und Bit-Images) müssen vom Standardformat ins gerätespezifische Format gewandelt werden. In der Anweisung switch (xtype) werden die benutzerdefinierten Objekte angepaßt. Die Breite und Höhe dieser Objekte wird etwas verkleinert. Damit ist es möglich, Boxen in aufeinanderfolgenden Zeilen zu plazieren, ohne daß sie sich direkt berühren (siehe SCRAP, Menü Optionen, Menüpunkt Einstellungen oder Abbildung der Druckerauswahlbox). Dadurch bekommt man wesentlich mehr Informationen auf dem Bildschirm unter.

Die beiden Zuweisungen

```
ob->ob_type = G_USERDEF; und
ob->ob_spec = (LONG)&check_blk; bzw. = (LONG)&radio_blk;
```

dürfen beim Laser C-Compiler nicht gemacht werden. Er kann leider noch keine benutzerdefinierten Objekte verwalten. Vielleicht klappt es in einer späteren Version. Alle anderen Operationen müssen aber gemacht werden, damit wenigstens die Größe der Box angepaßt wird.

- draw_checkbox
- draw_rbutton

Die beiden Funktionen draw_checkbox sowie draw_rbutton werden vom AES angesprungen, wenn die Checkbox bzw. ein runder Radio-Knopf gezeichnet oder verändert werden muß (siehe auch Kap. 2.5).

Damit sind die Erläuterungen zu SHOWGEM abgeschlossen. Die Teile für das Zeichnen von Metadateien und Bit-Image-Dateien sind auch im Programm SCRAP integriert. Noch

ein letzter Tip: Schauen Sie sich zuerst die Resource-Datei SHOWGEM.RSC und SHOWGEM.H an. Dann sehen Sie, welche Objekte einen Namen bekommen haben und so für das Programm wichtig sind.

Lassen Sie sich auch an den Programmstellen, wo Sie nicht alles verstehen, die beteiligten Variablen und Werte ausgeben (DEBUG = 1) und benutzen Sie dann die Funktion wait, um sich die Ergebnisse anzusehen (siehe auch show_debug).

# 5 Gestaltung von Benutzeroberflächen

Betrachtet man verschiedene GEM-Programme gleicher Kategorie (z.B. Textverarbeitung, Datenbanken) auf einem Atari ST, so fällt auf, daß einige einfach, andere wieder schwieriger oder sehr schwierig zu bedienen sind. Trotz gleicher Aufgabenstellung scheint es von mehreren Faktoren abzuhängen, daß Programme als „leicht" oder „schwer" erlernbar oder bedienbar gelten. GEM-Applikation ist offensichtlich nicht gleich GEM-Applikation, obwohl es in der Werbung immer wieder heißt „...ist eine GEM-Applikation und damit leicht zu bedienen."

Betrachtet man andere Rechner, wie den Macintosh, so sieht man eine Fülle von Programmen, die einem hohen Qualitätsstandard entsprechen. Daten von einem Programm können leicht in ein anderes Programm übernommen werden, und die meisten Programme sind leicht, zumindest aber ähnlich zu bedienen. Das liegt zum einen daran, daß beim Macintosh die grafische Benutzeroberfläche nicht auf dem Betriebssystem aufsetzt wie z.B. beim Atari ST (GEM auf TOS) oder dem IBM PC (GEM auf DOS). Zum anderen ist zu sagen, daß sich Programmierer an gewisse Richtlinien halten, die von der Bibel der Macintosh-Programmierung „Inside Macintosh" stammen.

Beim Atari beispielsweise gibt es eine Fülle von verschiedenen Grafikformaten. Jeder kocht – leger ausgedrückt – sein eigenens Süppchen (Doodle, Degas, Neochrome, STAD, Signum etc.). Um nun Grafik von einem Programm in ein anderes zu übernehmen, braucht man Konvertierungsprogramme oder Snap-Accessories. Dabei gibt es ähnlich wie beim Mac zwei von Digital Research eingeführte Standards: die Metadateien und die Bit-Image-Dateien. Erstere werden unter anderem von GEM-Draw, letztere von GEM-Paint erzeugt. Der Aufbau der beiden Dateien wurde in Kapitel 2 oder anderen Büchern wie z.B. „Software-Entwicklung auf dem Atari ST" genauestens erklärt.

Während beim bekannten Textverarbeitungsprogramm „1st Word Plus" von GST bei den ersten Versionen ein eigenes Format zur Abspeicherung von Bildern benutzt wurde, wird bei den neueren Versionen das Bit-Image-Datei-Format unterstützt. Damit können also Bilder aus GEM-Paint direkt in Word Plus übernommen werden. Dies ist insbesondere deshalb wichtig, weil Word Plus kein Grafikeditor ist.

Für das Design eines Programms, also Menüs, Dialogboxen, Bedienung etc. gibt es bis jetzt keine schriftlich festgelegten Standards auf dem ST. Würden die Automobilhersteller sich sowenig an Regeln halten wie Software-Entwickler auf dem ST, so kämen wohl Autos heraus, bei denen sich beispielsweise das Gaspedal links, das Kupplungspedal in der Mitte und das Bremspedal rechts befinden. Ein solches Auto würde niemand kaufen, wenn bereits alle anderen Autos die Pedale in der uns bekannten Reihenfolge anbieten würden.

Das Fehlen von Regeln und Standards in der Anfangsentwicklung bei Erscheinen des Atari ST führte wohl dazu, daß jeder nach eigenem Gutdünken programmierte und seine eigenen Regeln für die Besten hielt. Dennoch gibt es einige allgemeingültige Regeln, die man beachten sollte, damit GEM-Applikationen komfortabel in der Bedienung werden. Ein Teil dieser Regeln stammen aus der Arbeit von Card, Moran und Newell „The Psychology of Human-Computer Interaction". Es handelt sich bei den Autoren um Mitarbeiter von PARC (Palo Alto Research Center) von Xerox. Der Xerox Star war Anfang der achtziger Jahre der erste kommerzielle Rechner mit einer grafischen Benutzeroberfläche und einer Maus als Eingabegerät. Card und Moran arbeiteten am AIP-Projekt (Applied Information-Processing Psychology), also am Projekt für angewandte informationsverarbeitende Psychologie. Bevor nun Regeln für das Design von GEM-Programmen angegeben werden sollen, wird die Theorie das Wort haben. Wir werden sehen, auf welchen wissenschaftlichen Grundlagen die Regeln basieren, die später angeführt werden.

## 5.1 Theorie

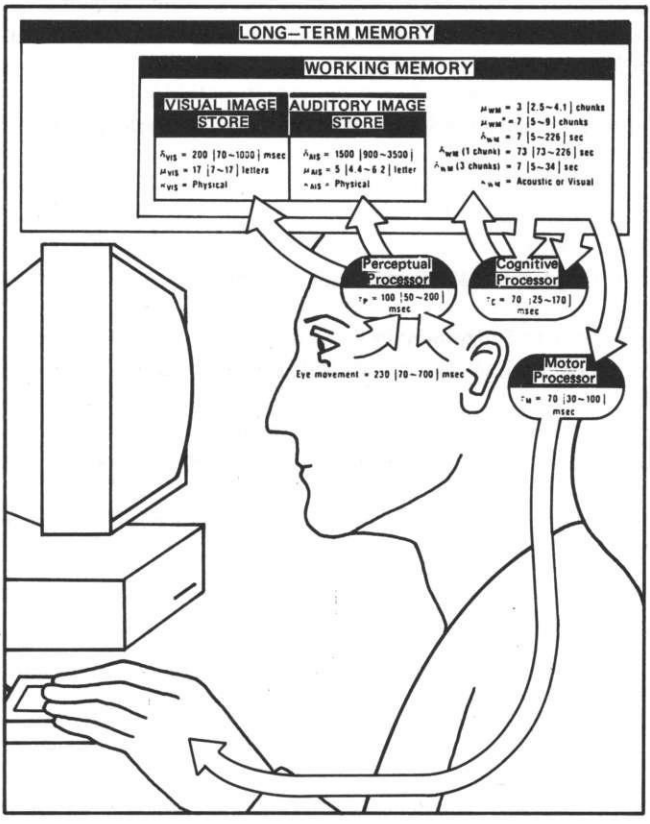
### 5.1.1 Die Mensch-Maschine-Schnittstelle

Trotz der Entwicklung der Schnittstellen von Lochstreifen über Lochkarten, alphanumerischen Terminals und den heutigen Grafik-Terminals mit Bedienung über Maus und Fenster ist es auffallend zu beobachten, daß viele unnötige Fehler in der Bedienung auftreten. Der Grund dafür ist zum einen darin zu suchen, daß die Interaktion mit dem Rechner sehr vielfältig sein kann. Maschinen, die früher von Menschen bedient wurden, hatten meist eine begrenzte Anzahl von Aufgaben zu erledigen, so daß der Bedeutungsbereich von Hebeln, Knöpfen und Rädern sehr eng war. Während Maschinen früher lediglich bedient wurden, treten die Menschen heute in echte Kommunikation mit der Maschine, d.h. mit dem Rechner.

Zum anderen führte die radikale Zunahme der Fähigkeiten eines Rechners sowie das verbesserte Preis/Leistungsverhältnis dazu, daß mehr Rechner für mehr Aufgaben eingesetzt werden konnten. Die Folge war eine große Anzahl neuer Stile in der Interaktion zwischen Mensch und Rechner. Damit verbunden waren komplett neu aufgebaute Schnittstellen, an die sich der Mensch jeweils gewöhnen mußte.

### 5.1.2 Das Modell „Mensch-als-Prozessor“

Um die Interaktion einer Person mit dem Rechner zu analysieren, wird der Mensch selbst als Prozessor betrachtet, welcher Informationen verarbeiten kann. (siehe Abb. 5.1).




Quelle: nach Cavet, Movan u. Newell The Psychology of Human-Computer Interactiar, S. 26

**Abbildung 5.1:** Speicher und Prozessoren des „Mensch-Prozessor-Modells“

Beim Betrachten des Modells erkennen wir: Es kann durch einen Satz von Speicher und Prozessoren sowie einer Menge von Prinzipien beschrieben werden und wird in drei interagierende Untersysteme zerteilt:

a) das Wahrnehmungssystem (Augen, Ohren)
b) das motorische System (Arme, Hände)
c) das Erkennungssystem (Gedächtnis, Gehirn)

Jedes dieser Systeme hat seine eigenen Speicher und Prozessoren. Das Wahrnehmungssystem besteht aus Sensoren und angeschlossenen Speicherpuffern wie dem Abbildungsspeicher für das Sehen und Hören. Es speichert die Daten des Sinnessystems, während

sie symbolisch decodiert werden. Das Erkennungssystem erhält symbolisch codierte Informationen aus dem Speicher des Kurzzeitgedächtnisses und benutzt Informationen aus dem Langzeitgedächtnis, um zu entscheiden, wie auf ein Ereignis reagiert werden soll. Die Reaktion wird dann vom motorischen System ausgeführt.

Es wird angenommen, daß jedes Untersystem seinen eigenen Prozessor und seinen eigenen Speicher besitzt, wobei die Prozessoren seriell oder quasiparallel, d.h. in der Art von Pipeline-Prozessoren arbeiten können.

Die Speicher und Prozessoren werden durch gewisse Parameter beschrieben. Die Parameter des Speichers sind:

- die Speicherkapazität in Einzelpunkten
- die Verfallszeit eines Einzelpunktes
- die Art der Verschlüsselung (physisch, akustisch, visuell, semantisch)

Die Speicherkapazität gibt an, wieviele Einzelpunkte der Prozessor sich merken kann. So ist der Speicher des Kurzzeitgedächtnisses kleiner als der des Langzeitgedächtnisses.

Die Verfallszeit gibt an, wann die Wahrscheinlichkeit, eine bestimmte Information aus dem Gedächtnis zurückholen zu können, weniger als 50% beträgt (Halbwertszeit).

- Die Art der Verschlüsselung gibt an, wie die Information letztendlich im Speicher vorliegt.

Der wichtigste Parameter eines Prozessors ist die Zykluszeit.

Sie gibt an, wie schnell Informationen aufgenommen werden oder Bewegungen durchgeführt werden können.

a) Das Wahrnehmungssystem

Das Wahrnehmungssystem führt Eindrücke aus der realen Welt, die vom Sensorsystem des Körpers aufgenommen werden, in interne Repräsentationen des Gehirns über. Ein gutes Beispiel ist das visuelle System. Die Retina des Auges reagiert auf Licht und „merkt" sich dessen Intensität, Wellenlänge und räumliche Verteilung. Obwohl das Auge eine Szenerie über einen breiten Winkel aufnimmt (fast 180 Grad), werden Details nur über einen vergleichsweise engen Winkel von ungefähr 2 Grad aufgenommen. Dieser Bereich wird auch Fovea genannt. Der Rest der Retina dient mehr der Orientierung. Das Auge ist beim Betrachten eines Objekts in ständiger Bewegung. Diese ruckartigen Bewegungen, auch Sakkaden genannt, nehmen jeweils eine Zeit von 30 ms in Anspruch, bevor die Augen auf den nächsten Punkt gesetzt werden. Sie verweilen dann dort für eine Dauer von 70 – 700 ms, was eine durchschnittliche Augenbewegungszeit von 230 ms ergibt.

Kurz nach dem Eintreffen des visuellen Reizes erscheint eine Repräsentation dieses Reizes im „Visuellen Abbildspeicher" (VIS = Visual Image Store). Ähnliches gilt für das Gehör (AIS = Acoustic Image Store). Die Information in diesen Speichern wird physisch als nicht-symbolisches Analogon zum externen Reiz gespeichert.

Zum Beispiel wird die Ziffer 2 nicht als solche gespeichert, sondern deren Repräsentation beinhaltet Dinge wie Krümmung und Länge. Experimente belegen die Tatsache des physischen Speicherns. So kann eine Testperson, welche eine farbige Liste von Zahlen vorgelegt bekommt, sehr schnell beispielsweise die grünen Zahlen herauspicken, während das Erkennen von geraden oder ungeraden Zahlen eine bestimmte Zeit erfordert. Im letzten Fall muß das Kurzzeitgedächtnis des Erkennungsprozessors zu Hilfe genommen werden. Für Buchstaben gilt eine Erkennungsrate von etwa 10 ms/Buchstabe.

Der visuelle Abbildspeicher hat eine Halbwertszeit von 200 ms, der akustische Abbildspeicher von 1500 ms. Dies bedeutet, daß nach der angegebenen Zeit die entsprechende Information nur noch schwer aus dem Speicher zurückzuholen ist.

Die Kapazität des visuellen Abbildspeichers beträgt 7 – 17 Buchstaben, die des akustischen Speichers 5 Buchstaben. Vor dem Verschwinden der Information muß diese ins Kurzzeitgedächtnis übertragen worden sein (s.u.).

Die Zykluszeit des Wahrnehmungsprozessors liegt bei etwa 100 ms, was bedeutet, daß Blinkraten auf dem Bildschirm eine gewisse Frequenz (10/s) nicht überschreiten sollten. Sonst werden sie nicht mehr unbedingt als diskrete Ereignisse wahrgenommen.

Dies gilt allerdings nicht, wenn der Reiz erhöht wird, was am Prinzip der variablen Wahrnehmungsprozessor-Rate deutlich wird: Die Zykluszeit des Wahrnehmungsprozessors variiert reziprok zur Intensität des Reizes.

b) Das motorische System

Für Rechnerbenutzer sind die besonders zu untersuchenden Systeme das Arm-Hand-System und das Kopf-Auge-System. Bewegungen laufen nicht kontinuierlich ab, sondern bestehen aus einer Reihe von Mikrobewegungen. Jede dieser Bewegungen benötigt etwa 70 ms. Dies kann als Zykluszeit des motorischen Prozessors angesehen werden.

Analysieren wir die Handbewegung eines Benutzers, welcher z.B. mit der Maus einen Pfeil auf dem Bildschirm in ein bestimmtes Gebiet bewegen muß (siehe Abb. 5.2).

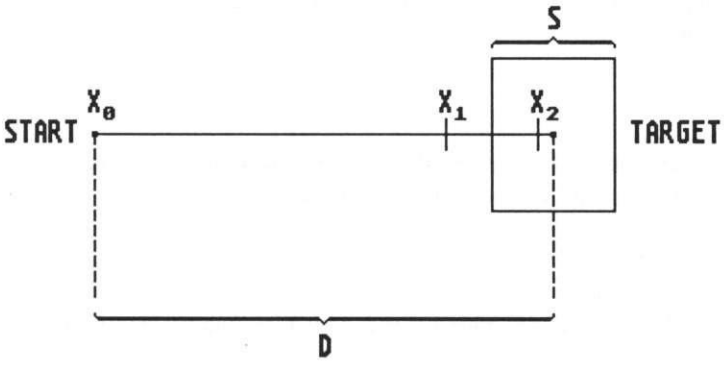



**Abbildung 5.2:** Analyse der Handbewegung eines Benutzers

Die Entfernung zum Zielpunkt sei D cm, wogegen das Zielrechteck S cm breit sei. Während die Hand nun auf das Zielquadrat zusteuert, führt sie eine Reihe von Mikrobewegungen aus. Nach jeder Bewegung werden vom Erkennungssystem die Hand beobachtet und eventuell Korrekturen herbeigeführt. Startpunkt sei $X_0$. Nach jedem neuen Schritt, bei dem die Hand nun näher an das Ziel kommt, wird der Korrekturfaktor kleiner. Mit Hilfe der Zykluszeiten der zugrundeliegenden Prozessoren kann man beweisen, daß die benötigte Zeit einzig und allein von der relativen Genauigkeit abhängt, d.h. vom Verhältnis zwischen Zieldistanz (D) und Zielgröße (S). Die Folgerungen werden weiter unten gezogen.

c) Das Erkennungssystem

Im einfachsten Fall dient das Erkennungssystem (kognitives System) dazu, Eingaben vom Wahrnehmungssystem in die richtigen Ausgaben an das Motorsystem umzusetzen. Die meisten von Menschen zu bewältigenden Aufgaben sind jedoch sehr komplex, so daß die Speicher des kognitiven Systems komplizierter sind als die der anderen Systeme.

Zwei wichtige Speicherbereiche finden sich im kognitiven System: das Kurzzeitgedächtnis (WM = Working Memory), welches Informationen enthält, die in der aktuellen Situation gebraucht werden, und das Langzeitgedächtnis (LTM = Long Term Memory), welches Wissen speichert, das auch später noch benutzt werden kann.

Das Kurzzeitgedächtnis

Es enthält Zwischenprodukte von Gedanken und die Repräsentationen, die vom Wahrnehmungssystem produziert werden. Es bildet die Register des kognitiven Prozessors. Dabei handelt es sich im Prinzip um eine Untermenge der Elemente des Langzeitgedächtnisses, welche gerade aktiviert ist. Die Codierung im Kurzzeitgedächtnis geschieht mittels akustischer oder visueller symbolischer Codes. Das Kurzzeitgedächtnis wird nicht von physischen Parametern wie etwa der Intensität eines Reizes beeinflußt.

Im Kurzzeitgedächtnis bilden sich Symbole. Die Ersetzung der Symbole hängt nun vom Inhalt des Langzeitgedächtnisses ab und damit von der Person und der zu lösenden Aufgabe. So kann die Buchstabenfolge

BMIACRFSAB

von nur wenigen Menschen nach kurzem Anblick wiederholt werden (versuchen Sie es einmal), während dieselbe Liste mit etwas geänderter Reihenfolge

IBMRCABASF

leichter behalten werden kann, wenn sie etwa beim Lesen oder Buchstabieren in

IBM RCA BASF

aufgeteilt wird. Statt neun Symbole wie oben hat man sich nur noch drei Symbole zu merken (eine Computerfirma, einen Medienkonzern und einen Chemie-Konzern). Dies gilt natürlich nur, wenn die drei Symbole eine Bedeutung für den jeweiligen Menschen haben. Daraus kann man folgern, daß das Kurzzeitgedächtnis erheblich entlastet wird, wenn in der Kommunikation zwischen Mensch und Rechner Symbole benutzt werden, die möglichst vielen Menschen bekannt sind.

Das Kurzzeitgedächtnis kann nur eine begrenzte Zahl von Symbolen aufnehmen. Wenn neue Symbole hinzukommen, interferieren sie mit den bereits vorhandenen Symbolen im Kurzzeitgedächtnis. Die Folge ist das Vergessen der „älteren" Symbole. Die Geschwindigkeit, mit der die Symbole vergessen werden, hängt natürlich von vielen Faktoren ab, wie z.B. der Anzahl von anderen Symbolen, die sich der Benutzer merken muß.

Muß man sich nur ein Symbol merken, so kann man sich an dieses nach etwa 73 Sekunden noch aus dem Kurzzeitgedächtnis erinnern. Bei drei Symbolen beträgt die Halbwertszeit nur noch 7 Sekunden. Wird die reine Kapazität des Kurzzeitgedächtnisses zugrundegelegt, so kommt man auf eine Anzahl von drei Symbolen, welche man sich dort merken kann. Diese reine Kapazität wird allerdings erhöht, da das Langzeitgedächtnis mit benutzt wird (siehe obiges Beispiel mit den Buchstaben). Dann ergibt sich eine Kapazität von 7 (+/− 2) Symbolen. Dies wurde bereits 1956 von dem Psychologen George A. Miller herausgefunden. Wir werden diese Erkenntnis später verwenden, wenn es um Menüs geht.

Das Langzeitgedächtnis

Es enthält die Masse des verfügbaren Wissens eines Benutzers und besteht aus einem Netzwerk von Symbolen, die in einer bestimmten Beziehung zueinander stehen. Auf sie wird über den Inhalt des Kurzzeitgedächtnisses zugegriffen. Der Inhalt des Langzeitgedächtnisses besteht aus der deklarativen und prozeduralen Wissensbasis. Im Prinzip gibt es kein Vergessen im Langzeitgedächtnis, so daß die Halbwertszeit als unendlich angesehen werden kann.

Die Rückholung von Information hängt jedoch davon ab, ob bestimmte Assoziationen gefunden werden können. Zwei Gründe kann es dafür geben, daß keine Information mehr aus dem Langzeitgedächtnis verfügbar ist:

1. Effektive Assoziationen zur Wiedergewinnung können nicht gefunden werden.

2. Ähnliche Assoziationen mit mehreren Symbolen interferieren bei der Rückholung des Zielsymbols.

Die Verschlüsselung von Information im Langzeitgedächtnis liegt in semantischer Form vor, d.h. die neuen Informationen werden verknüpft mit Informationen, die bereits im Gedächtnis stehen. Ein gutes Beispiel sind (gute oder schlechte) Vorurteile.

Das Prinzip der spezifischen Verschlüsselung besagt, daß sie auf alle wahrgenommenen Dinge angewandt wird. Sie entscheiden, wie etwas im Langzeitgedächtnis gespeichert wird und damit, welche Wiedergewinnungsstrategien angewandt werden, um die gespeicherte Information zu erhalten. Es kommt also darauf an, mit welchen Assoziationen eine bestimmte Information verbunden wird, um sie wieder aus dem Langzeitgedächtnis zu holen.

Ein Beispiel: Ein Benutzer legt mehrere Dateien auf einem Inhaltsverzeichnis an. Alle diese Dateien bezeichnen Möbelstücke. Eine davon heißt „Bank", wobei er z.B. an eine Eckbank denkt. Später untersucht er das Inhaltsverzeichnis, um alle seine Dateien anzusehen. Im Inhaltsverzeichnis befinden sich nun viele Dateien, die er nicht angelegt hat. Als sein Blick auf die Datei „Bank" fällt, assoziiert er damit ein Geldinstitut, weil er noch einige Dinge auf seiner Bank zu erledigen hat. Die Datei „Bank" wird er nicht als seine erkennen, da er diese mit einem Geldinstitut, nicht mit einem Möbelstück assoziiert.

Durch die Überlagerung mit anderen Symbolen im Gedächtnis, die stärker durch bestimmte Assoziationen aktiviert wurden, kann es vorkommen, daß Information, obgleich physisch vorhanden, verlorengeht. Man nennt dies auch das Unterscheidungsprinzip. Die Schwierigkeit, etwas aus dem Gedächtnis wiederzugewinnen, hängt von der Anzahl der „Kandidaten" ab, die für die entsprechende Assoziation in Frage kommen.

An einmalige und besondere Situationen kann man sich also leichter erinnern als an alltägliche Dinge, die immer wieder auftreten. Außerdem ist es eher möglich, Informationen aus dem Gedächtnis wiederzugewinnen, je mehr Assoziationen mit dieser Information verbunden sind. Wenn ein Benutzer sich später an eine Sache erinnern möchte, so ist die beste Strategie diejenige, diese Sache mit etwas in Verbindung zu bringen, was schon im Langzeitgedächtnis existiert, wobei die Verbindung neu sein sollte, damit sie nicht mit anderen interferiert.

Ähnlich wie bei einem Rechner der Fetch-Execute-Zyklus arbeitet der Erkennungs-Aktions-Zyklus beim Menschen. Der Inhalt des Kurzzeitgedächtnisses aktiviert Teile des Inhalts des Langzeitgedächtnisses (Erkennen), die wiederum den Inhalt des Kurzzeitgedächtnisses modifizieren (Aktion), was den Zustand für den nächsten Zyklus festlegt. Der Erkennungsprozessor hat dabei eine Zykluszeit von 70 ms. Sie variiert allerdings und hängt unter anderem von der gestellten Aufgabe ab. Die Zeit kann durch Übung, verminderte Anforderung an Genauigkeit oder größere Bemühungen des Benutzers verkürzt werden.

Dies beendet die einführende Beschreibung des Modells „Mensch-als-Prozessor". Zusammenfassend ist zu sagen: Das Modell besteht aus einer Menge interagierender Speicher und Prozessoren und einer Menge von Speichercodierungstypen. Jeder Prozessor hat eine bestimmte Zykluszeit, die jeweils etwa eine zehntel Sekunde beträgt.

Ein solch einfaches Modell wird natürlicherweise nicht der Reichhaltigkeit und der Feinheit des menschlichen Geistes gerecht. Aber es hilft uns wenigstens etwas dabei, die menschliche Durchführung von Aufgaben zu verstehen, vorherzusagen oder sogar zu berechnen, zumindest was die Mensch-Maschine-Interaktion angeht.

### 5.1.3 Weitere Gesetze und Prinzipien

Bei der Durchführung von Aufgaben agiert der Mensch nach bestimmten Gesetzen, die hier dargelegt werden sollen. Daraus können Folgerungen für die Implementierung effizienter Mensch-Maschine-Schnittstelllen abgeleitet werden. Der Einfachheit halber werden die Formeln mit Worten umschrieben.

a) Das Gesetz der Übung

Die Zeit, eine gestellte Aufgabe zu lösen, verringert sich exponentiell mit der Anzahl der Versuche. Dieses Prinzip wird auch das Lernprinzip des Wahrnehmungs- und Motor-Prozessors genannt. Jeder von uns hat dies wohl schon erlebt, wenn immer gleiche wiederkehrende Worte auf einer Tastatur eingegeben werden müssen. Die Eingabe dieser Worte geht mit der Zeit immer schneller vonstatten. Dies gilt zumindest bei einem ungeübten Benutzer, der mit Hilfe des Ein-Finger-Suchsystems arbeitet.

Ein Beispiel: Ein Schaltpult hat zehn Lämpchen, unter denen sich zehn Tasten befinden. Je nachdem, welches Lämpchen aufleuchtet, muß der Benutzer die entsprechende Taste drücken. Wenn er für den 1000sten Versuch beispielsweise 1,48 Sekunden benötigt und für den 2000sten Versuch 1.15 Sekunden, so kann man sich ausrechnen, wie lange er für den 50000sten Versuch benötigen wird. Es sind dies nach der Formel für das Exponentialgesetz 0,37 Sekunden. Experimente haben diese Werte bestätigt.

Das Exponentialgesetz der Übung spielt eine große Rolle bei der Arbeit an der Tastatur und hat mehrere praktische Konsequenzen (s.u.).

b) Das Unsicherheitsprinzip

Muß der Anwender eine Wahl zwischen verschiedenen Alternativen treffen, so erhöht sich die Reaktionszeit, d.h. die Zeit bis zum Treffen der Entscheidung, mit der Anzahl der Alternativen.

Der Grund dafür ist die Zykluszeit des kognitiven Prozessors (70 ms), die für jede Alternative anfällt. Allerdings verwertet man die Alternativen meist in Gruppen. Man entscheidet sich zuerst für eine Gruppe, dann für eine Untergruppe usw. Durch diese fortgesetzte Zerteilung (z.B. Halbierung) ergibt sich keine lineare Gleichung, sondern eine logarithmische (Hick's Gesetz).

Etwas anders sieht es aus, wenn die Alternativen nicht gleichgewichtig sind, d.h. wenn sie eine unterschiedliche Auftrittswahrscheinlichkeit haben.

Ein Beispiel: Ein Telefon-Verteiler hat 10 Knöpfe. Wenn eines der Lämpchen an den Knöpfen angeht, so muß eine Sekretärin den Knopf drücken und den Anruf beantworten. Wie groß ist der prozentuelle Unterschied in der Reaktionszeit, wenn (1) jedes der Lämpchen gleich häufig aufleuchet und (2) zwei der Lämpchen zu 50% und 40% aufleuchten und der Rest zu 10%?

Nach dem Gesetz von Hick wird für Fall 1 eine Unsicherheit von 3,46 ausgerechnet und für Fall 2 von 2,14. Als prozentueller Unterschied ergibt sich 2,14/3,46 = 62%. Das bedeutet, daß in Fall 2 gegenüber Fall 1 die Sekretärin nur 62% der Zeit benötigt.

Das Prinzip sollte beim Design der Menüauswahl und von Dialogboxen einer GEM-Applikation berücksichtigt werden, da es Einfluß auf die Reaktionszeit des Benutzers hat.

c) Vergessen von gerade erhaltener Information

Je nachdem, wo Information aktuell gespeichert ist, erreicht die Halbwertszeit, wie oben gesehen, Werte von 200 ms (Wahrnehmungsspeicher), 7000 ms (Kurzzeitgedächtnis) und unendlich (Langzeitgedächtnis) oder, wenn die Verhältnisse gekürzt werden, 1:35:unendlich. Dies bedeutet, daß die Wiedergewinnungsstrategie davon abhängt, zu welchem Zeitpunkt die Information aufgenommen wurde. Dies entscheidet dann, aus welchem Gedächtnis die Information geholt werden kann.

Für die Wiedergewinnung von Information, die einige Sekunden nach der Speicherung passiert, kann das Kurz- oder das Langzeitgedächtnis herangezogen werden (oder beides). Nach einigen Minuten kann allerdings nur noch das Langzeitgedächtnis „befragt" werden.

Beispiel: Einem Programmierer werden verbal ein Dutzend Dateinamen genannt, die er in sein System laden soll. Wie soll der Programmierer unter der Annahme, daß alle Namen verschieden sind und nicht aus sinnlosen Silben bestehen (also nicht XSDFGRO etc.), die Dateinamen aufschreiben, um möglichst selten nachfragen zu müssen?

Da die Kapazität des Kurzzeitgedächtnisses mit zwölf Symbolen überlastet ist, wird er einige Namen vergessen. Beim Versuch, sich zu erinnern, wird aber das Kurzzeitgedächtnis belastet, was ein weiteres Vergessen bewirkt. Die Lösung besteht darin, daß der Programmierer zuerst versuchen sollte, die zuletzt genannten Namen aufzuschreiben. Diese befinden sich noch im Kurzzeitgedächtnis. Die erstgenannten befinden sich – wenn überhaupt – nur noch im Langzeitgedächtnis.

Natürlich hängt dieses Beipiel stark von den Namen der Dateien ab. Haben diese beispielsweise Namen wie INIT1, INIT2, ..., INIT12, so wird kein Mensch Probleme haben, all diese aus dem Gedächtnis aufzuschreiben, da er sich nur zwei Symbole merken muß: INIT und die Regel für die Numerierung (bis einschließlich 12).

Noch ein Beispiel: Wie lange kann ein Benutzer verweilen, um den Dateinamen CAT vollständig zu schreiben, im Gegensatz zum Dateinamen TXD?

Da TXD keinen Sinn macht, besteht der Name aus drei Symbolen (Buchstaben). Drei Symbole verschwinden im Schnitt nach 7 Sekunden aus dem Kurzzeitgedächtnis. Bei nur einem Symbol (CAT wird z.B. mit Katze assoziiert) sind es 73 Sekunden, so daß die Verweilzeit das zehnfache beträgt.

Auch das letzte Beispiel zeigt, daß es das Kurzzeitgedächtnis erheblich entlastet, wenn bedeutungsvolle Namen gewählt werden. Dies umso mehr, da jene auch leichter ins Langzeitgedächtnis aufgenommen werden können.

d) Interferenz im Kurzzeitgedächtnis

Interferenz bedeutet Überlagerung (hier durch Ähnlichkeit). Werden Symbole im Kurzzeitgedächtnis mit anderen Symbolen überlagert, so wird es schwieriger, die Symbole zu trennen und damit wiederzugewinnen (siehe auch das Unterscheidungsprinzip). Die Ähnlichkeit von zwei Symbolen im Gedächtnis hängt von der geistigen Repräsentation der Symbole ab. Diese wiederum hängt von dem Gedächtnis ab, in dem sich das Symbol befindet. Die beiden wichtigsten Überlagerungsformen sind akustische (visuelle) und semantische Interferenz. Symbole im Kurzzeitgedächtnis sind gewöhnlicherweise anfälliger für akustische bzw. visuelle Interferenz (sie werden gerne mit anderen Symbolen verwechselt, die ähnlich klingen bzw. aussehen), da das Kurzzeitgedächtnis gewöhnlicherweise akustische bzw. visuelle Codierung benutzt. Symbole im Langzeitgedächtnis sind anfälliger für semantische Interferenz, da das Langzeitgedächtnis semantische Codierung benutzt.

Ein Beispiel: In einem System wird die Fehlerbehandlung auf folgende Weise gelöst. Bei einem Systemabsturz erscheinen bis zu fünf Codewörter zu je drei Buchstaben auf einem speziellen Bildschirm, die der Operator niederschreiben (auf Papier übertragen) muß. Jedes der Wörter – es gibt insgesamt sehr viele solcher Wörter – hat eine spezielle Bedeutung und stellt eine spezielle mnemotechnische Abkürzung dar, die man sich leicht merken kann. Was sollte eher vermieden werden, Codewörter, die ähnlich klingen oder Codewörter, die eine ähnliche Bedeutung haben?

Lösung:
Da die Codewörter sofort niedergeschrieben werden müssen, werden sie sich zum größten Teil im Kurzzeitgedächtnis befinden. Da sich das Kurzzeitgedächtnis akustischer Codierung bedient, werden Übertragungsfehler vor allem wegen akustischer Interferenzen zwischen den Codewörtern auftreten. Daraus folgt, daß ähnlich klingende Codewörter vermieden werden sollten.

e) Interferenz im Langzeitgedächtnis

Das Unterscheidungsprinzip (s.o.) sagt aus, daß die Schwierigkeit, sich an etwas Bestimmtes zu erinnern, davon abhängt, an welch andere Dinge man sich erinnert, die durch die gleichen Zusammenhänge aus dem Gedächtnis geholt werden.

Ein Beispiel: Ein Benutzer ist dabei, die Bedienung eines neuen, z.B. zeilenorientierten Texteditors zu erlernen. Dieser ist nahezu identisch mit einem Editor, den er bereits kennt. Dessen Kommandos haben aber eine andere Syntax bei gleicher Bedeutung (z.B. „ERASE“ statt „DELETE“ für das Löschen einer Zeile). Wird das Lernen der Bedienung des neuen Editors mit der Fähigkeit des Benutzers, sich an die Kommandos des alten Editors zu erinnern, interferieren?

Ja. Wenn der Benutzer die Bedienung des neuen Editors lernt, gibt es neue Symbole im Gedächtnis, die denen des alten Editors ähneln. Nach der Tatsache des Unterscheidungsprinzips interferieren diese mit Methoden, die alten Befehle aus dem Gedächtnis zurückzuholen. In der Tat ist es eine allgemeine Erfahrung für Programmierer, unfähig zu sein, sich zu erinnern, wie das alte System zu benutzen war, obwohl sie Hunderte von Stunden damit gearbeitet hatten, nachdem sie ein ähnliches neues System gelernt hatten.

Bei menügesteuerten Systemen ist dieses Problem nicht so groß, da die Alternativen aufgezeigt werden. Im einen Editor mag das Menü „ERASE" heißen, im anderen „DELETE". Da aber immer nur das jeweilige Menü angezeigt wird, gibt es keine Verwechslung. Anders ist dies bei Befehlen, die nicht als Menüs auftreten, so bei bildschirmorientierten Editoren. In einem Editor bedeutet die Taste „Pfeil rechts" zusammen mit „Shift" das Springen ans Ende der Zeile, bei anderen Editoren muß die Taste „Control" statt „Shift" gedrückt werden.

Noch ein Wort zum Prinzip, wie Information aus dem Langzeitgedächtnis geholt wird. Information wird bei jedem Zyklus des kognitiven Prozessors aus dem Gedächtnis geholt. Dieses „Wiederholen" der Information ist jedoch nicht immer erfolgreich. Ist jedoch genügend Zeit verfügbar, um das Gedächtnis zu untersuchen, so können bestimmte Strategien benutzt werden, um das Langzeitgedächtnis immer wieder zu untersuchen. Ein gutes Beispiel ist das Erinnern an ein bekanntes, aber selten benutztes Kommando.

Wir wollen uns einmal vor Augen halten, wie schwierig es für einen Benutzer ist zu versuchen, eine Tatsache aus seinem Langzeitgedächtnis abzurufen (Prinzip der spezifischen Verschlüsselung, s.o.). Wenn der Benutzer eine Tatsache (Symbol, Merkmal, etc.) lernt, wird es in irgendeiner Form verschlüsselt im Gedächtnis abgelegt. Die Verschlüsselung beinhaltet verschiedene Hilfsmöglichkeiten, sich später daran zu erinnern. Zum Zeitpunkt des „Sich Erinnerns" kennt der Benutzer weder die Tatsache noch die Hilfsmöglichkeiten. Er muß deswegen raten oder vermuten. Dabei werden Hilfsmöglichkeiten ins Kurzzeitgedächtnis gebracht. Dort dienen sie beim nächsten Zyklus als „Ruf" ins Langzeitgedächtnis. Die Vermutungen können nun gut sein und sofort zum Erfolg führen. Wenn nicht, so können sie immerhin Informationen freilegen, die dann beim nächsten Versuch zum Erfolg führen können.

Bei einem Versuch wurden Versuchspersonen sieben Jahre nach dem Abitur gefragt, an welche Mitabiturienten sie sich erinnern können. Nach zehn Stunden des Versuchs, sich zu erinnern, konnten die Versuchspersonen immer noch Namen aus dem Langzeitgedächtnis zurückholen. Ihre Strategie war folgende: In Gedanken suchten die Personen nach Gesichtern, die sie auf früheren Parties gesehen hatten, gingen das Alphabet durch, schlenderten vertraute Straßen herunter und fragten nach den Hausbewohnern. Je mehr Zeit sie hatten, an desto mehr Namen erinnerten sie sich. Natürlich führte dies nicht dazu, daß sich die Personen an alle Mitabiturienten erinnerten. Dies kann man an einer Kurve, deren Absizze die Stunden und deren Ordinate die Anzahl der Mitabiturienten angeben, beobachten. Sie nähert sich asymptotisch einem Grenzwert.

f) Das Rationalitätsprinzip

Ein Anwender versucht, seine Ziele dadurch zu erreichen, daß er die Dinge tut, die tatsächlich benötigt werden, um die Aufgabe selbst zu lösen. Ein Großteil der Komplexität des menschlichen Verhaltens läßt sich nicht auf die Komplexität des Menschen als solchen zurückführen (er versucht lediglich, seine Aufgabe zu lösen), sondern auf die Komplexität der Umgebung, in der die Aufgabe gestellt wird. Daraus folgt, daß, um das menschliche Verhalten sowohl zu verstehen als auch einigermaßen vorhersehen zu können, die Aufgabe analysiert werden muß. Damit werden die verschlungenen Pfade des rationellen Verhaltens entdeckt. Wir kommen daher zu dem Phänomen, das man auch das Fundamentalprinzip der Aufgabenanalyse nennt.

Das Rationalitätsprinzip: Ein Benutzer versucht, eine gestellte Aufgabe durch rationale Aktionen zu lösen, wobei die Struktur der Aufgabe und die Informationen, die er erhält, gegeben sind. Eingegrenzt wird sein Handeln von seinem Wissen und seinen Fähigkeiten, d.h:

> Ziele + Aufgabe + Operatoren + Eingabeinformation + Wissen + Fähigkeiten -> Verhalten.

Das Prinzip bietet eine Menge von Formulierungen an, die benutzt werden können, um das Verhalten einer Person voraussagen zu können. Die ersten drei Faktoren (Ziel, Aufgabe, Operatoren) geben die objektive Situation an, die anderen zeigen etwas versteckte Zwänge auf, nämlich was der Benutzer erkennt, was er weiß und wie er etwas berechnen kann.

g) Das Prinzip des Problemraumes

Rationelles Verhalten kann oft präziser beschrieben werden. Angenommen, eine Person hat das Ziel, ein Theorem mit Hilfe der Regeln der symbolischen Logik zu beweisen. Dann gibt es eine Menge von mentalen Zuständen, durch die sie geht. Beschreibbar sind sie in Termen von symbolischen Ausdrücken. Außerdem gibt es eine Anzahl von Operatoren, mit deren Hilfe man von einem Zustand in einen anderen kommt. Diese Menge von Zuständen und dazugehörigen Operatoren nennt man auch den Problemraum.

Das Prinzip des Problemraums: Das rationale Lösen eines Problems kann auf folgende Art beschrieben werden: Es existieren

- eine Menge von Wissenszuständen
- Operatoren, die einen Zustand in einen anderen überführen
- Regeln der anzuwendenden Operatoren
- Wissen um die Entscheidung, welcher Operator jeweils als nächstes benutzt wird

Es gibt verschiedene Problemräume für verschiedene Aufgaben, und es gibt auch Änderungen in einem Problemraum mit der Zeit, wenn der Benutzer mehr Erfahrung über die Struktur der Aufgabe erlangt.

Ein Beispiel ist das Lösen eines verschlüsselten arithmetischen Puzzles. Dabei wird sowohl der Verstand als auch eine Trial-And-Error-Methode benutzt.

```
   D O N A L D
+  G E R A L D
-------------
   R O B E R T        D = 5
```

Die obigen Buchstaben sollen durch Ziffern von 0 bis 9 ersetzt werden. Gleiche Buchstaben bedeuten gleiche Ziffern. Unterschiedliche Buchstaben repräsentieren unterschiedliche Ziffern. Das Ergebnis soll eine mathematisch korrekte Addition sein. D ist vorgegeben als 5.

Es sind:

Zustände: Zuweisungen von Ziffern an Buchstaben

Operatoren:

- „Weise zu" (einem Buchstaben eine Ziffer)
- „Untersuche eine Spalte"
- „Erzeuge mögliche Ziffern" (von Buchstaben)
- „Teste Ziffer"

Regeln: D + D = T etc.

Während der Lösung des Problems kann sich der Problemraum ändern, da mehr Wissen über die gestellte Aufgabe zusammengetragen wird. Es wird nun wichtig sein, die Menge der Zustände und Operatoren ausreichend klein zu halten.

## 5.2 Folgerungen

Die oben ausgeführten theoretischen Abhandlungen sind die Grundlage der Folgerungen, die für die Implementierung von Software unter einer grafischen Benutzeroberfläche wie GEM getroffen werden können. Es hat sich gezeigt, daß beim Einhalten bestimmter Regeln die Benutzerschnittstelle effizienter wird. Durch den Versuch, das Verhalten des Benutzers vorauszusehen, werden Entscheidungen erleichtert, die er zu treffen hat.

Die Benutzerschnittstelle muß schnell und konsistent sein, um den Anwender nicht von der eigentlichen Arbeit abzulenken.

Die Folgerungen beziehen sich zwar beispielmäßig auf die grafische Benutzeroberfläche GEM bzw. X/GEM, die auf einem ATARI ST/TT oder MS-DOS- bzw. FlexOS-Rechner zur Verfügung steht. Sie gelten aber im Prinzip auch für andere (grafische) Benutzeroberflächen.

### 5.2.1 Menüs

Die Kapazität des Kurzzeitgedächtnisses begrenzt die Anzahl der Wahlmöglichkeiten, die ein Benutzer auf einen Blick erfassen kann. Die Anzahl sollte, wie schon oben ausgeführt, niemals die Zahl sieben übersteigen. Wird dieses Gesetz verletzt, dann muß der Anwender mehrere Male auf die Menüauswahl blicken und dabei pausieren, um sich zu überlegen, welche Auswahl er treffen soll.

Kann die Anzahl der Menüs nicht verringert werden, so sollten zumindest logische Untergruppen von Menüs gebildet werden. Diese Untergruppen werden dann vom Benutzer wie ein Symbol erkannt, so daß sich die Kapazität erhöht. Auf dieselbe Weise geht das menschliche Gehirn vor, das sich einen Satz weniger über die einzelnen Buchstaben, sondern mehr über die Wörter in diesem Satz merken kann.

Die Menüzeile selbst, die von GEM zur Verfügung gestellt wird, ist ein gutes Beispiel für Symbolgruppen. Unter den einzelnen Menüs verbergen sich wiederum Menüpunkte. Würden alle Menüpunkte auf einmal angezeigt, dann würde das Kurzzeitgedächtnis überlastet werden.

Als Beispiel kann hier der Menüeintrag „File“ des Textsystems Wordplus herangezogen werden. Er unterteilt sich in 5 Untergruppen, die jeweils logisch zusammengehören (Abb. 5.3). Dort kann aber noch mehr abgelesen werden: Menüeinträgen, bei denen beim Anwählen eine Dialogbox folgt, sollten 3 Punkte folgen (z.B. „Öffnen...“). Der Benutzer kann sich dann schon darauf einstellen, daß in der Mitte des Bildschirms eine Dialogbox erscheinen wird, indem er sich mit der Maus vom Menüeintrag in die Bildschirmmitte bewegt, während diese aufgebaut wird. Diese 3-Punkte-Regel wird übrigens von Tim Oren empfohlen, der das GEM maßgeblich mitentwickelt hat.

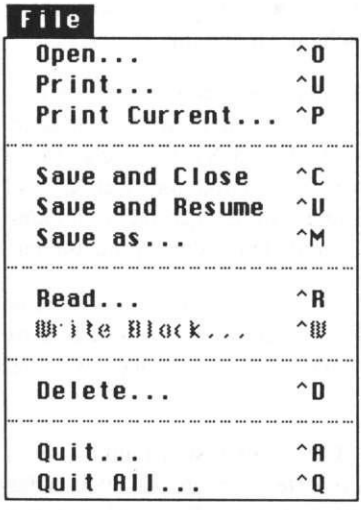



**Abbildung 5.3:** „File“-Menüpunkt in Wordplus 3.14

Ein schlechtes Beispiel ist der Aufbau des Menüs „Text" bei Tempus 2.0 (Abb. 5.4). Es ist total überladen (13 Menüpunkte, davon 12 hintereinander). Außerdem bestehen die einzelnen Menüpunkte aus zu vielen Buchstaben so daß zu viel Gesamttext im Drop-Down-Menü steht.

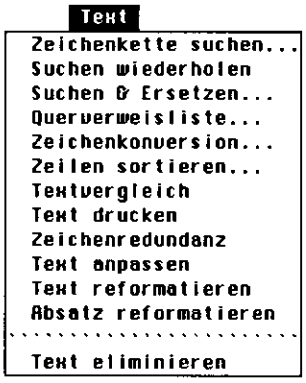



**Abbildung 5.4:** „Text"-Menü in Tempus 2.0

Ein Benutzer, der daraus einen Menüpunkt auswählen möchte, muß ständig fast den kompletten Text eines Drop-Down-Menüs lesen, da das erste Wort zum Großteil redundant ist. Es handelt sich meist um „Text". So heißt aber auch das Menü. Würde man im Text-Menü nur solche Punkte unterbringen, die tatsächlich mit dem Wort „Text" beginnen, so könnte man sich dieses Wort wiederum sparen und die Menüpunkte wären (in der Horizontalen) kürzer. Außerdem befinden sich im Menüpunkt „Speziell" noch die Punkte „Text expandieren" und Text komprimieren". Assoziiert nun ein Benutzer seine Tätigkeit „Text komprimieren" verständlicherweise mit dem Oberbegriff „Text", so sucht er vergeblich im Menü (Text), welches nicht den gewünschten Menüpunkt bietet.

Menüs sollten einen Standardaufbau haben, damit sich ein Benutzer beim Starten des Programms sofort „heimisch" fühlt. Das erste Menü von links (Desk) auf dem Atari ist dasjenige mit den Deskaccessories. Bei neueren GEM-Versionen (2.X, 3.X) steht dieses Menü ganz rechts und hat als Titel immer den Namen der aktuellen Applikation. Bei X/GEM gibt es keine Accessories, da beliebige X/GEM-Programme im Hintergrund laufen können. Anstelle der Accessories treten dann die Namen der gerade laufenden Applikationen.

Wenn man mit vielen Programmen arbeitet, die teilweise gleiche Menüs aufweisen, so weiß man mitunter auf den ersten Blick nicht, wie das Programm heißt, das gerade aktiv ist. Bei den neueren GEM-Versionen einschließlich X/GEM ist dies, wie oben erwähnt, kein Problem.

Beim ST gibt es nun Programme, welche nur einfach „Desk" als ersten Punkt in ihrer Menüzeile haben, andere benutzen das Atari-Symbol. Am besten wird jedoch auch in diesen Programmen der Name ohne Suffix in die Menüzeile gebracht. Dann erkennt man sofort, wie das aktive Programm heißt.

Zu dem Menü ist nur soviel zu sagen, daß die Applikation möglichst so sauber programmiert wird, daß Deskaccessories zugelassen werden. Es gibt auch Programme, die die Deskaccessories ausblenden (Signum, STAD). Deskaccessories haben selbst keine Menüzeile. Bei großen Accessories ist dies bisweilen aber wünschenswert, ebenso bei Programmen, die wahlweise als Applikation und als Accessory laufen können. Damit man die Bedienung eines solchen Accessories beibehalten kann, wird in Kapitel 6 eine Routine vorgestellt, die eine Menüzeile für ein Accessory in ein Fenster legt. Dadurch kann eine Applikation, die als Accessory im Hintergrund läuft, praktisch genauso bedient werden wie das Programm. Es muß kein Byte im Programm umgeschrieben werden, damit dies funktioniert. Die vorgestellten Routinen erledigen dies automatisch.

Der zweite Menütitel sollte immer „Datei" oder „File" heißen. In ihm sollten sich Menüpunkte befinden, mit denen die Ein/Ausgabe geregelt wird. Beispiele sind der GEM Desktop und das Resource Construction Set. (s. Abb. 5.5).

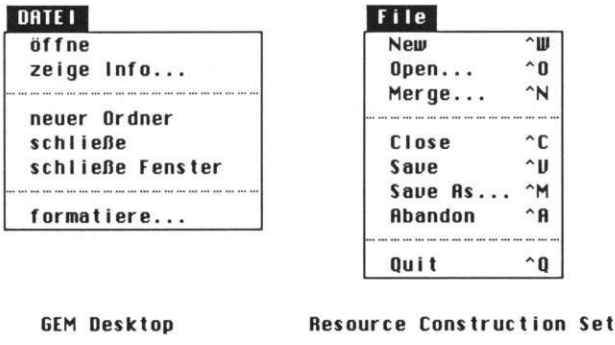



**Abbildung 5.5:** Datei-Menüs des GEM Desktop und des Resource Construction Sets.

Zu den Menüpunkten sollten möglichst immer gehören:

Neu (New): Ein neues Dokument (Resource-Datei, etc.) wird geöffnet. Einen Namen bekommt das Dokument dann beim erstmaligen Speichern. Kann kein neues Dokument geöffnet werden, muß der Menüpunkt abgeschaltet (grau) sein. Auf diesen Menüpunkt kann verzichtet werden, wenn beim Öffnen (s.u.) bereits ein Name gewählt wird, der vorher nicht da war.

Öffnen... (Open...): Öffnet ein existierendes Dokument (Resource-Datei, Programm etc.) oder ein Objekt, welches selektiert ist. Bei Bedarf erscheint die Datei-Auswahl-Box, in welcher der Benutzer eine Datei auch auf einem anderen Laufwerk auswählen darf.

Schließen (Close): Schließt das oberste Fenster und ein damit eventuell zusammenhängendes Dokument (Resource-Datei, etc). Wurde eine Änderung an dem durch das Fenster repräsentierten Objekt gemacht, so muß dem Benutzer die Gelegenheit gegeben werden,

das Schließen abzuwenden. Dieselben Aktionen laufen ab, wenn der Benutzer in das Schließkästchen klickt. Wird ein Fenster geschlossen, ohne daß der Text verlorengeht, so kann eine Sicherheitsabfrage zunächst entfallen. Es muß dann aber die Möglichkeit bestehen, das entsprechende Fenster wieder zu öffnen. Dazu muß das Fenster in ein Icon umgewandelt werden. Dieses kann dann durch Anklicken wieder geöffnet werden. Ein Beispiel hierzu ist das in Kapitel 6 vorgestellte Demoprogramm, aber auch z.B. TEMPUS.

Info... (Info...): Zeigt eine spezifische Information an. Ist ein Objekt selektiert, so wird dessen Information angezeigt, ansonsten Information über das aktive Fenster. Ist kein Fenster aktiv (offen), so wird Programminformation angezeigt.

Hilfe... (Help...): Zeigt einen spezifischen Hilfstext an. Ist ein Objekt selektiert, so wird dessen Hilfstext angezeigt, ansonsten Hilfstext über das aktive Fenster. Ist kein Fenster bzw. nur der Desktop aktiv (offen), so wird allgemeiner Hilfstext über das Programm angezeigt. Dieser Menüpunkt kann eventuell entfallen, wenn ein eigenes Hilfe-Menü existiert (wie z.B. bei Turbo C).

Sichern (Save): Sichert ein Dokument (eine Resource-Datei, eine Grafik, etc.). Bei neuen Dokumenten wird zuerst noch nach einem Namen gefragt, da dieser noch nicht vergeben ist. Als Alternative kann auch der Menüpunkt abgeschaltet und nur ein „Sichern als..." erlaubt sein. Nach dem Sichern bleibt das Dokument geöffnet.

Sichern als... (Save as...): Sichert eine Kopie des aktiven Dokuments. Dabei erscheint die Datei-Auswahl-Box, über die der Benutzer den Namen eingeben kann. Nach dem Beenden der Funktion sollte sich der Name des Dokuments (Fenster-Titel) ändern, so daß ab nun der neu eingegebene Name beim nächsten Sichern berücksichtigt wird.

Drucken... (Print...): Druckt ein Dokument oder das aktive Fenster. Vorher können in einer Dialogbox druckerspezifische Parameter wie z.B. Anzahl der Kopien etc. eingestellt werden.

Datei löschen... (Delete File...): Optional kann ein Menüpunkt vorgesehen werden, um eine Datei zu löschen. Dies gilt vor allem für Applikationen, die Backups von Dokumenten auf Diskette speichern und somit die Möglichkeit geben wollen, diese zu löschen, falls das Original nicht mehr auf die Diskette paßt.

Disk formatieren... (Format Disk...): Das Formatieren von Disketten aus einem Programm soll vor allem dazu dienen, auftretende Probleme beim Abspeichern von Dokumenten auf gefüllte Datenträger zu umgehen. Man stelle sich eine Situation vor, in der ein wichtiges Dokument nicht mehr abgespeichert werden kann, weil keine Diskette mit genügend Speicherplatz zur Verfügung steht. Muß man dann das Programm verlassen, um eine Diskette zu formatieren, so geht das Dokument verloren. Der Menüpunkt kann entfallen, wenn das Formatieren auch bei Ablauf eines anderen Programms möglich ist. Dies ist z.B. bei FlexOS der Fall, wo man jederzeit einen anderen Prozeß starten und von dort formatieren kann.

Beenden (Quit): Der letzte Menüpunkt sollte immer „Beenden" oder „Verlassen" lauten. Wurden Dokumente geändert, so sollte der Benutzer dazu aufgefordert werden, diese vorher abzuspeichern.

Der dritte Menütitel sollte immer „Bearbeiten" (Edit) lauten. In ihm befinden sich unter anderem die Befehle:

Undo: Die letzte Änderung wird rückgängig gemacht.

Ausschneiden (Cut): Der vorher ausgewählte Block wird ausgeschnitten und aus dem Original entfernt. Der Block wird vorher in einen Zwischenspeicher gerettet, um ein „Undo" oder ein „Einfügen" möglich zu machen.

Kopieren (Copy): Wie „Ausschneiden", aber ohne Löschen des Originals.

Einfügen (Paste): Der vorher ausgeschnittene oder kopierte Teil wird an der aktuellen Cursorposition eingefügt. Die Operation kann mehrere Male hintereinander ausgeführt werden.

Löschen (Clear): Wie „Ausschneiden", jedoch ohne Speicherung des Blocks.

Alles auswählen (Select all): Das gesamte Dokument wird ausgewählt. Dies entspricht z.B. bei Textdateien einem Blockanfang am ersten Buchstaben der ersten Zeile, einem Blockende am letzten Buchstaben der letzten Zeile.

Zusätzlich können sich Befehle zum Speichern auf eine Zwischenablage (Klemmbrett oder Clipboard) in diesem Menü befinden (s. Kap. 5.5). In unserer Beispielapplikation werden dazu die gleichen Menüs verwendet. Ein Schalter („Auf GEM-Klemmbrett") gibt dann an, ob die Operationen im schnellen Arbeitsspeicher oder auf dem langsamen, aber residenten GEM-Klemmbrett stattfinden.

Beim Einsatz der Icon-Technik können einige der o.g. Menüpunkte auf Icon-Funktionen zurückgeführt werden. Angenommen, man hat Standard-Icons wie Papierkorb, Drucker, Diskette, Klemmbrett sowie Icons für Dokumente (s. Abb. 5.6).

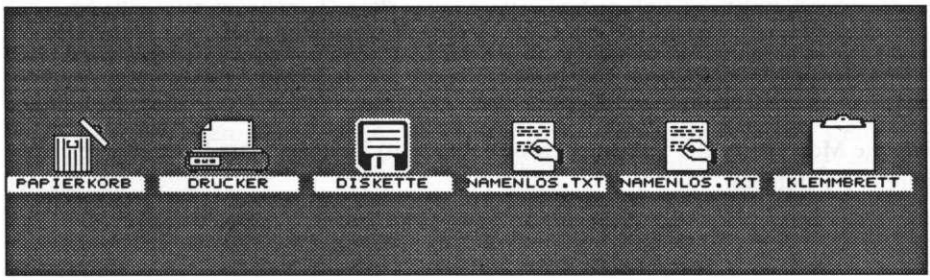



**Abbildung 5.6:** Standard-Icons

Dann ist es möglich, durch Modifizieren der Icons bestimmte Funktionen aufzurufen.

Öffnen: Alle Icons sollten geöffnet werden können. Der Papierkorb zeigt die zuletzt gelöschten Inhalte, der Drucker zeigt eine Druckereinstellungsbox, die Diskette den Disketteninhalt, das Klemmbrett das GEM-Clipboard und die Dokumente deren Inhalt.

Folgende Operationen können noch möglich sein:

Ziehen der Diskette:

- auf den Papierkorb: Löschen einer Datei
- auf den Drucker: Drucken einer Datei
- auf das Klemmbrett: Kopieren einer Datei ins GEM-Clipboard
- auf das Dokument: Laden einer Datei

Ziehen des Klemmbretts:

- auf den Papierkorb: Löschen des GEM-Clipboards
- auf den Drucker: Drucken des GEM-Clipboards
- auf die Diskette: Kopieren des GEM-Clipboards in eine Datei
- auf das Dokument: Einlesen des GEM-Clipboards in ein Dokument

Ziehen eines Dokuments:

- auf den Papierkorb: Löschen des Dokuments aus dem Speicher
- auf den Drucker: Drucken des Dokuments
- auf die Diskette: Sichern des Dokuments (Sichern als...)
- auf das Klemmbrett: Sichern des Dokuments in das GEM-Clipboard

Durch die vielfältigen Möglichkeiten des Ziehens eines Objekts auf ein anderes können viele Menüpunkte eingespart werden.

Um die Koordination des motorischen Systems zu entlasten (Handbewegung zwischen Tastatur und Maus), sollten alle Menüpunkte zusätzlich über Tastenkombinationen aufzurufen sein. Der Benutzer kann dann wählen, wie er ein Menü lieber aufrufen möchte. Dabei sollten sich aber alle Software-Entwickler an einen gemeinsamen Standard halten. Im Prinzip gibt es schon eine Art Standard, der aber nirgends dokumentiert ist. Dies wird hiermit nachgeholt.

In jedem Menüpunkt soll sich der zugehörige Tastencode am rechten Rand desselben befinden. An der Art der Darstellung kann der Benutzer auf die Tastenkombination schließen. Es können auftreten (siehe Abb. 5.7):

a) Normalzeichen (Großbuchstaben, Sonderzeichen)
b) Buchstaben mit gedrückter Control-Taste
c) Buchstaben mit gedrückter Alternate-Taste
d) Funktionstasten (Normal, Shift, Control oder Alternate)

**Abbildung 5.7:** Menüs mit Tastaturanzeige

Bei jedem Menüpunkt sollte versucht werden, den Anfangsbuchstaben zu verwenden. Dies wird nicht immer gelingen, so daß auch andere Buchstaben aus dem Kontext herangezogen werden können. Für die Standardfunktionen CUT, COPY, PASTE und UNDO sollten die Kombinationen (X, C, V und Z) benutzt werden, da dies in vielen Programmen schon auf diese Weise realisiert wird. Als Kombination bietet sich Control oder Alternate an. Die Control-Kombination hat sich als Standard bereits herauskristallisiert. Da jedoch Control-C bei X/GEM – zumindest unter FlexOS – immer ein vorzeitiges Beenden des Programmes erwirkt, könnte auch auf Alternate-Kombinationen ausgewichen werden, um die Kompatibilität zwischen den Versionen zu erhalten.

Da für internationale Versionen die Anfangsbuchstaben der Menüpunkte meist anders lauten, gibt es im Prinzip zwei Möglichkeiten: Man behält die Tastenkombinationen bei oder ändert sie entsprechend des Textes, der für das jeweilige Land maßgebend ist.

Entwickler werden nun sicher einwenden, daß es mühsam ist, für jedes Land andere Kombinationen zu benutzen, da diese im Programm entsprechend abgefangen werden müssen. Keine Sorge! In den Funktionen, die in Kapitel 6 angeboten werden, befindet sich eine Routine (is_menu_key), die für eine vorgegebene Tastenkombination und einen vorgegebenen Menübaum die Nummer des Menüpunktes ermittelt und zurückliefert.

Dies bedeutet, daß die Vergabe von Tastenkombinationen von außen erfolgen kann, nämlich durch Angabe im Menübaum in der Resource-Datei! Bei Änderung der Tastenkombinationen durch Übersetzung in andere Sprachen ergibt sich somit kein Zusatzaufwand für den Entwickler. Man könnte sogar zur Laufzeit die Menüpunkte entsprechend ändern, was bedeutet, daß es auch keinen Aufwand darstellt, wenn der Benutzer sich die Tastenkombinationen selbst definieren dürfte.

### 5.2.2 Mausbewegungen

Aus dem Gesetz von Fitt (s.o.) lassen sich eine Menge von Informationen zur Geschwindigkeitssteigerung der Benutzerschnittstelle ableiten. Die Objekte, die im Arbeitsbereich liegen, sollten sich eigentlich in der Nähe der Bildschirmmitte befinden, da die Maus dann im Durchschnitt eine kleinere Strecke zurückzulegen hat, um Objekte auszuwählen. Da die Bildschirmmitte nicht unendlich groß und oft verdeckt ist und zudem nicht alle Objekte dort Platz haben, sollten zumindest diejenigen Objekte, die semantisch zusammengehören, auch zusammen plaziert werden können.

Am Beispiel des Aufbaus des Desktop im Datenbanksystem ADIMENS ST soll obiges Gesetz veranschaulicht werden. Dort können die Symbole für die einzelnen logischen Dateien einer Datenbank, sowie die Funktionssymbole wie Papierkorb, Drucker, Diskette etc. beliebig plaziert werden. Die Stellungen der Objekte können abgespeichert werden, so daß bei jedem Neustart der gleiche Desktop erscheint. In Abbildung 5.8 sind die Symbole für die Ausgabe in einer Gruppe zusammengefaßt, während der Papierkorb, der zum Löschen von Datensätzen dient (gefährlich), abseits plaziert wurde.

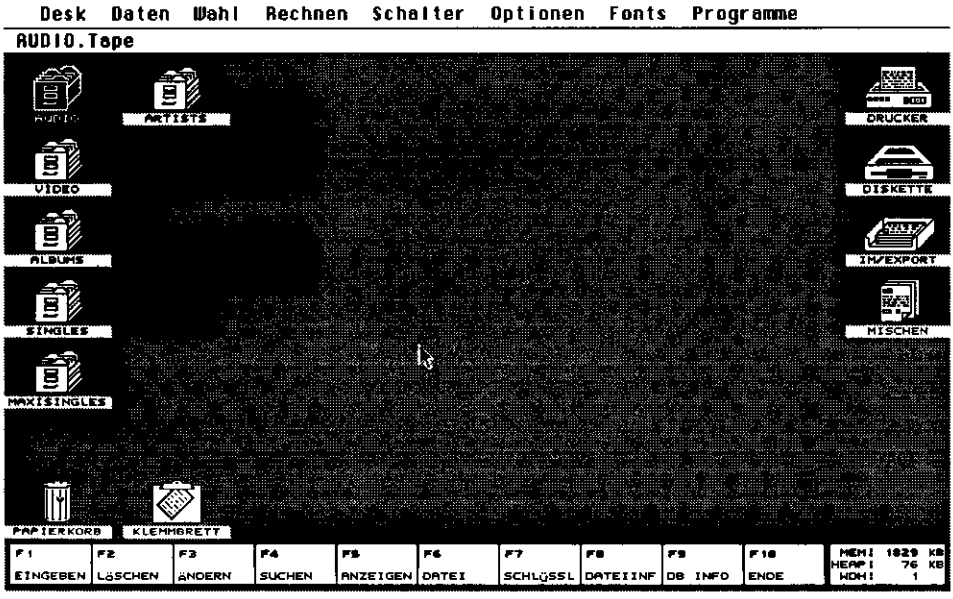



**Abbildung 5.8:** Der Desktop des Datenbankausführungsteils EXEC von ADIMENS ST

Ist nun eine große Menge von gleichen Operationen durchführbar, so kann das Funktions-Symbol (z.B. das Import/Export-Symbol) in die Nähe der Objekte plaziert werden, auf die die Funktion angewandt werden soll. Dadurch können die Operationen schneller durchgeführt werden, da die Mauswege kurz sind.

Die Kästchen (Knöpfe), in denen bei Dialogboxen mit Hilfe der Maus Optionen und Funktionen ausgewählt werden können, dürfen nicht zu klein sein, damit eine ausreichende Treffsicherheit gewährleistet ist. Außerdem sollte ein ausreichender Zwischenraum zwischen den Funktionskästchen sein, damit in den Randregionen kein Positionierungsfehler auftreten kann. In Abbildung 5.9 sind die Knöpfe „OK", „HILFE" und „ABBRUCH" der oberen Dialogbox um einiges größer als bei der unteren Dialogbox. Außerdem haben die Knöpfe der unteren Dialogbox einen zu kleinen Abstand zueinander.

**Abbildung 5.9:** Knöpfe in einer Dialogbox

Eine weitere Möglichkeit, Mauswege zu optimieren, sind sogenannte Pop-Up-Menüs. Sie erscheinen an der Stelle, an der mit dem Mauszeiger geklickt wird (siehe Abb. 5.10).

Dies bedeutet, daß die Vergabe von Tastenkombinationen von außen erfolgen kann, nämlich durch Angabe im Menübaum in der Resource-Datei! Bei Änderung der Tastenkombinationen durch Übersetzung in andere Sprachen ergibt sich somit kein Zusatzaufwand für den Entwickler. Man könnte sogar zur Laufzeit die Menüpunkte entsprechend ändern, was bedeutet, daß es auch keinen Aufwand darstellt, wenn der Benutzer sich die Tastenkombinationen selbst definieren dürfte.

### 5.2.2 Mausbewegungen

Aus dem Gesetz von Fitt (s.o.) lassen sich eine Menge von Informationen zur Geschwindigkeitssteigerung der Benutzerschnittstelle ableiten. Die Objekte, die im Arbeitsbereich liegen, sollten sich eigentlich in der Nähe der Bildschirmmitte befinden, da die Maus dann im Durchschnitt eine kleinere Strecke zurückzulegen hat, um Objekte auszuwählen. Da die Bildschirmmitte nicht unendlich groß und oft verdeckt ist und zudem nicht alle Objekte dort Platz haben, sollten zumindest diejenigen Objekte, die semantisch zusammengehören, auch zusammen plaziert werden können.

Am Beispiel des Aufbaus des Desktop im Datenbanksystem ADIMENS ST soll obiges Gesetz veranschaulicht werden. Dort können die Symbole für die einzelnen logischen Dateien einer Datenbank, sowie die Funktionssymbole wie Papierkorb, Drucker, Diskette etc. beliebig plaziert werden. Die Stellungen der Objekte können abgespeichert werden, so daß bei jedem Neustart der gleiche Desktop erscheint. In Abbildung 5.8 sind die Symbole für die Ausgabe in einer Gruppe zusammengefaßt, während der Papierkorb, der zum Löschen von Datensätzen dient (gefährlich), abseits plaziert wurde.

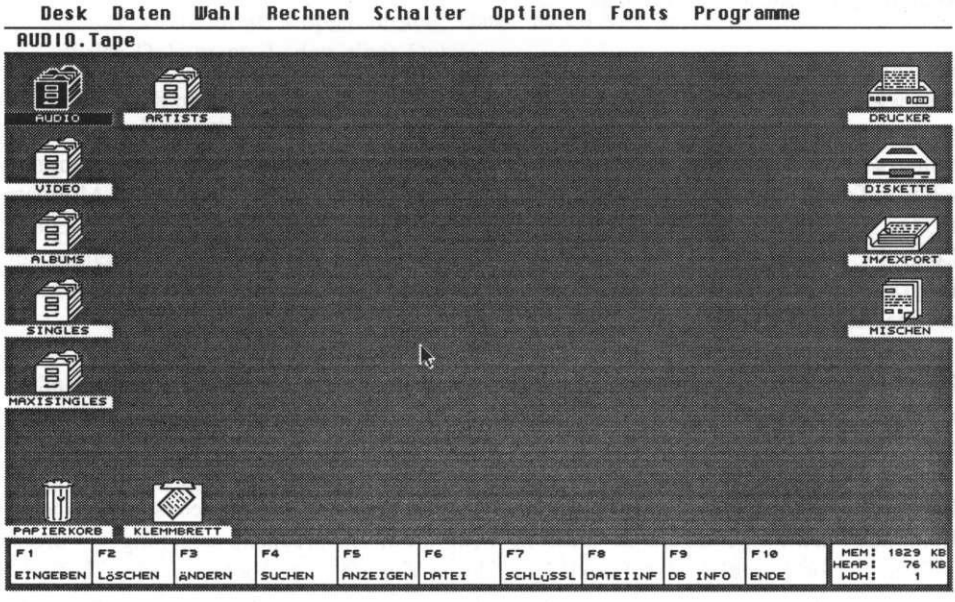



**Abbildung 5.8:** Der Desktop des Datenbankausführungsteils EXEC von ADIMENS ST

Ist nun eine große Menge von gleichen Operationen durchführbar, so kann das Funktions-Symbol (z.B. das Import/Export-Symbol) in die Nähe der Objekte plaziert werden, auf die die Funktion angewandt werden soll. Dadurch können die Operationen schneller durchgeführt werden, da die Mauswege kurz sind.

Die Kästchen (Knöpfe), in denen bei Dialogboxen mit Hilfe der Maus Optionen und Funktionen ausgewählt werden können, dürfen nicht zu klein sein, damit eine ausreichende Treffsicherheit gewährleistet ist. Außerdem sollte ein ausreichender Zwischenraum zwischen den Funktionskästchen sein, damit in den Randregionen kein Positionierungsfehler auftreten kann. In Abbildung 5.9 sind die Knöpfe „OK", „HILFE" und „ABBRUCH" der oberen Dialogbox um einiges größer als bei der unteren Dialogbox. Außerdem haben die Knöpfe der unteren Dialogbox einen zu kleinen Abstand zueinander.

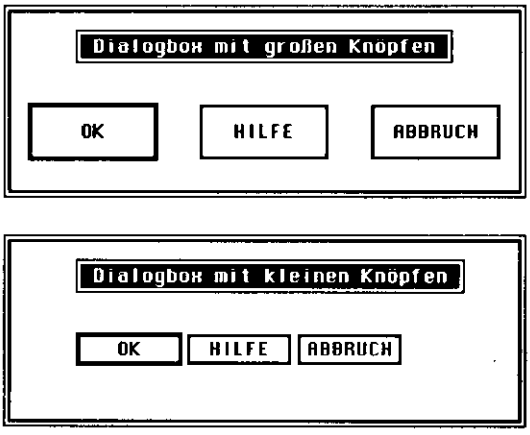



**Abbildung 5.9:** Knöpfe in einer Dialogbox

Eine weitere Möglichkeit, Mauswege zu optimieren, sind sogenannte Pop-Up-Menüs. Sie erscheinen an der Stelle, an der mit dem Mauszeiger geklickt wird (siehe Abb. 5.10).

Dies bedeutet, daß die Vergabe von Tastenkombinationen von außen erfolgen kann, nämlich durch Angabe im Menübaum in der Resource-Datei! Bei Änderung der Tastenkombinationen durch Übersetzung in andere Sprachen ergibt sich somit kein Zusatzaufwand für den Entwickler. Man könnte sogar zur Laufzeit die Menüpunkte entsprechend ändern, was bedeutet, daß es auch keinen Aufwand darstellt, wenn der Benutzer sich die Tastenkombinationen selbst definieren dürfte.

### 5.2.2 Mausbewegungen

Aus dem Gesetz von Fitt (s.o.) lassen sich eine Menge von Informationen zur Geschwindigkeitssteigerung der Benutzerschnittstelle ableiten. Die Objekte, die im Arbeitsbereich liegen, sollten sich eigentlich in der Nähe der Bildschirmmitte befinden, da die Maus dann im Durchschnitt eine kleinere Strecke zurückzulegen hat, um Objekte auszuwählen. Da die Bildschirmmitte nicht unendlich groß und oft verdeckt ist und zudem nicht alle Objekte dort Platz haben, sollten zumindest diejenigen Objekte, die semantisch zusammengehören, auch zusammen plaziert werden können.

Am Beispiel des Aufbaus des Desktop im Datenbanksystem ADIMENS ST soll obiges Gesetz veranschaulicht werden. Dort können die Symbole für die einzelnen logischen Dateien einer Datenbank, sowie die Funktionssymbole wie Papierkorb, Drucker, Diskette etc. beliebig plaziert werden. Die Stellungen der Objekte können abgespeichert werden, so daß bei jedem Neustart der gleiche Desktop erscheint. In Abbildung 5.8 sind die Symbole für die Ausgabe in einer Gruppe zusammengefaßt, während der Papierkorb, der zum Löschen von Datensätzen dient (gefährlich), abseits plaziert wurde.

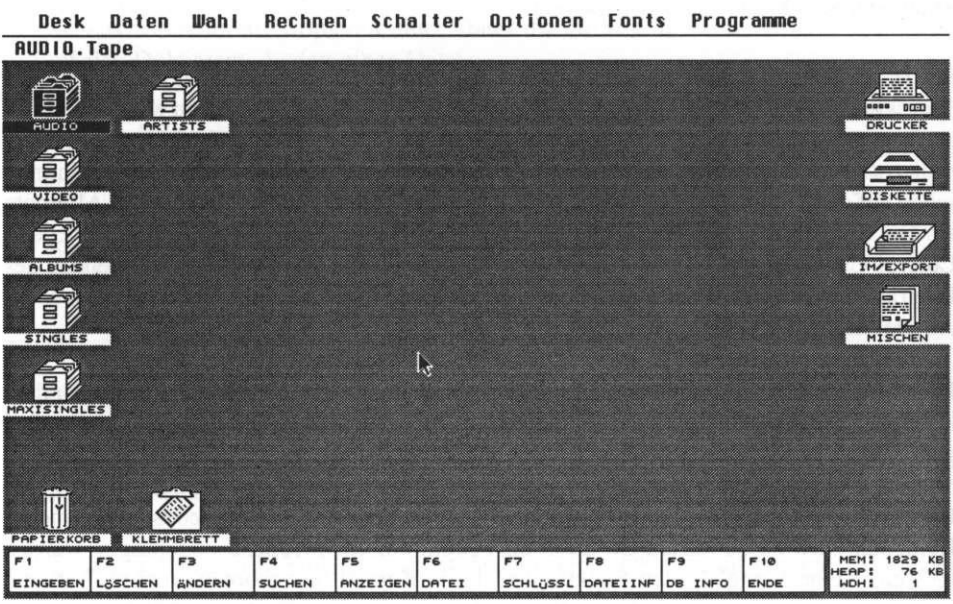



**Abbildung 5.8:** Der Desktop des Datenbankausführungsteils EXEC von ADIMENS ST

Ist nun eine große Menge von gleichen Operationen durchführbar, so kann das Funktions-Symbol (z.B. das Import/Export-Symbol) in die Nähe der Objekte plaziert werden, auf die die Funktion angewandt werden soll. Dadurch können die Operationen schneller durchgeführt werden, da die Mauswege kurz sind.

Die Kästchen (Knöpfe), in denen bei Dialogboxen mit Hilfe der Maus Optionen und Funktionen ausgewählt werden können, dürfen nicht zu klein sein, damit eine ausreichende Treffsicherheit gewährleistet ist. Außerdem sollte ein ausreichender Zwischenraum zwischen den Funktionskästchen sein, damit in den Randregionen kein Positionierungsfehler auftreten kann. In Abbildung 5.9 sind die Knöpfe „OK", „HILFE" und „ABBRUCH" der oberen Dialogbox um einiges größer als bei der unteren Dialogbox. Außerdem haben die Knöpfe der unteren Dialogbox einen zu kleinen Abstand zueinander.

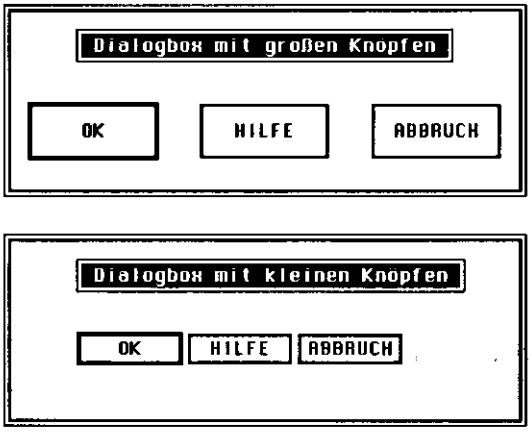



**Abbildung 5.9:** Knöpfe in einer Dialogbox

Eine weitere Möglichkeit, Mauswege zu optimieren, sind sogenannte Pop-Up-Menüs. Sie erscheinen an der Stelle, an der mit dem Mauszeiger geklickt wird (siehe Abb. 5.10).

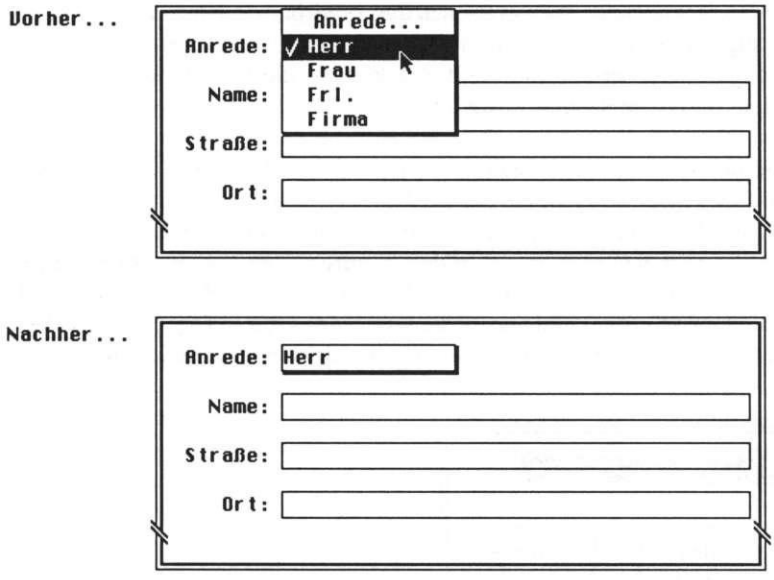



**Abbildung 5.10:** Pop-Up-Menü „Anrede"

Das Menü springt hoch und zeigt eine Menge von Alternativen, aus denen nun ausgewählt werden kann. Pop-Up-Menüs bieten sich vor allem an, wenn es eine begrenzte Menge von Alternativen gibt, wobei es eine starke Überforderung des Kurzzeitgedächtnisses wäre, alle Alternativen auf einmal zu listen. Entwickler müssen sich keine Sorge um die Programmierung von Pop-Up-Menüs machen, da die dazu nötigen Routinen in Kapitel 6 vorgestellt werden.

Es erhebt sich die Frage, wie Pop-Up-Menüs zu bedienen sind. Beim Apple Macintosh ergibt sich kein Problem, da dies wie bei normalen Menüs geschieht: Anklicken und bei gedrückten Knopf auswählen, dann loslassen. Bei GEM geschieht ein Auswählen von Menüs ohne gedrückten Knopf. Das Aktivieren des Pop-Up-Menüs muß aber durch Klicken stattfinden. Wie geschieht nun die weitere Bedienung?

Nach dem Aktivieren durch Klick gibt es zwei Möglichkeiten. Bleibt der Mausknopf gedrückt, so kann eine Alternative ausgewählt und danach der Mausknopf losgelassen werden (Macintosh-like). Wird der Mausknopf nach dem Aktivieren sofort wieder losgelassen (Einfachklick wie zum Selektieren von Objekten), so ist der Mauskopf nun oben, und es muß auf einen erneuten Klick gewartet werden (GEM-like). Die Pop-Up-Menüs sind nun so flexibel implementiert, daß sich der Benutzer heraussuchen kann, welche Methode er lieber verwenden möchte. Dazu muß an den bestehenden Routinen nichts geändert werden. Je nach Aktivieren des Menüs reagiert dieses entsprechend.

### 5.2.3 Übung

Je häufiger ein Anwender ein System benutzt, desto mehr Übung bekommt er im Umgang mit ihm. Nach einer bestimmten Zeit weiß er, in welchen Menüs sich bestimmte Kommandos verbergen, oder wohin er zur Beantwortung eines Dialogs mit der Maus fahren muß. Hinter diesem Aspekt steckt das Exponentialgesetz der Übung (s.o.).

Dialogboxen sollten sich immer in der Mitte des Bildschirms öffnen, damit sie einen Platz einnehmen, der schwerlich übersehen werden kann. Dasselbe gilt für Warn- oder Fehlermeldungen. Um einen Dialog zu beenden oder einen Fehler zu quittieren, muß ein bestimmtes Kästchen (Knöpfe) in der Dialogbox angeklickt werden. Die Kästchen sollten sich immer am unteren Ende der Dialogbox befinden, so daß der Anwender mit der Maus schon während des Aufbaus derselben in diese Region fahren kann.

Bei manchen GEM-Programmen stehen diese Knöpfe alle links oder rechts, oben oder sogar in der Mitte der Dialogbox. Wechselt man so von einem Programm zum anderen, muß der Benutzer sich immer wieder umorientieren.

Wir legen fest: Es sollte sich immer eine Mindestanzahl von Knöpfen in einer Dialogbox befinden. Position, Reihenfolge und Beschriftung dieser Standardknöpfe sollte bei allen Programmen gleich aussehen (siehe auch Abb. 5.11).

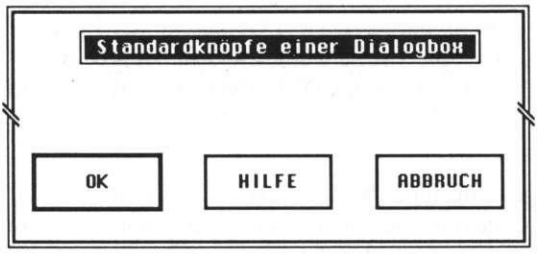



**Abbildung 5.11:** Standard-Knöpfe in einer Dialogbox.

Die Knöpfe sollten von links nach rechts, falls dieser Platz nicht ausreicht, ausnahmsweise auch von oben nach unten folgendermaßen beschriftet werden:

OK: Der Dialog wird velassen. Die Einstellungen werden übernommen bzw. ausgeführt.

HILFE: Der Benutzer bekommt Information über die Bedeutung der Dialogbox und wie sie zu handhaben ist (siehe auch 5.3.). Nach Erscheinen der Hilfebox befindet man sich wieder im Dialog.

ABBRUCH: Der Dialog wird verlassen. Die Einstellungen bzw. Änderungen werden nicht übernommen.

Bei Dialogboxen, die nur Information anzeigen, kann auf ABBRUCH verzichtet werden. Meistens ist dann auch HILFE unnötig.

Ein weiteres Kriterium, sich die Übung des Benutzers zunutze zu machen, sind die schon oben erwähnten Menüs mit den Tastaturkombinationen wie CUT, COPY, PASTE (X-C-V). Diese drei Tasten liegen nicht zufällig nebeneinander!

### 5.2.4 Aufmerksamkeit

Um die Aufmerksamkeit des Anwenders auf sich zu ziehen, kann das Prinzip der variablen Wahrnehmungsprozessor-Rate herangezogen werden. Wie aus 5.1.2 hervorgeht, müssen Ereignisse, die als getrennt angesehen werden sollen, in einem bestimmten zeitlichen Abstand voneinander auftreten. Dieser Abstand wird durch die Zykluszeit des visuellen Prozessors bestimmt. Durch Änderung der Intensität des Reizes kann die Rate beeinflußt werden.

Jeder von uns kennt wohl schon verschiedene Arten, wie man die Aufmerksamkeit des Benutzers auf sich ziehen kann. Für das akustische System kann die Benutzung von Geräuschen angebracht sein. Bei schwerwiegenden Fehlern kann die Fehlermeldung durch ein Geräusch, wie z.B. einen Pieps begleitet werden. Dies lenkt die Aufmerksamkeit des Benutzers auf den Bildschirm, wenn aus seinem Lautsprecher das Geräusch ertönt. Der Pieps sollte aber unbedingt abschaltbar sein, da durch zu viele Geräusche der Benutzer verunsichert wird. In einem Großraumbüro beispielsweise würden andere Mitarbeiter gestört werden oder sie würden sich über den Benutzer des Dialogsystems lustig machen („Macht der heute aber einen Haufen Fehler...“).

Für das visuelle Wahrnehmungsvermögen vor allem beim Auftreten von Fehlern sorgt im Prinzip schon die Gestaltung der Alertboxen. Sie haben jeweils ein großes Icon am linken Rand, welches beim Erscheinen des Fehlers oder der Warnung ins Auge sticht.

Eine Möglichkeit, Aufmerksamkeit auf das Programm zu ziehen, wird auch durch Farbe erreicht. Allerdings sollten nicht zu viele Farbtöne verwendet werden, da sonst der Reiz verlorengeht. Gut sind Komplementärfarben, also Schwarz/Weiß, Rot/Grün oder Blau/Gelb.

### 5.2.5 Vorhersehbarkeit, Modi und Transparenz

Eine rationelle Person arbeitet nach dem Prinzip des Problemraums (siehe 5.1.3), d.h. sie ändert die Zustände ihres Problemraums, bis sie den End- oder Zielzustand erreicht hat. Der Anfangszustand könnte in einem Datenbanksystem eine leere Datenbank sein, der Endzustand eine gefüllte Datenbank mit Adressen von Kunden.

Das Verhalten des Benutzers kann durch das Rationalitätsprinzip erklärt werden. Es scheint, daß der typische Anwender eines Programms eine Menge Zeit damit verbringt, sich zu überlegen, was er als nächstes tun soll. Eines der Schlüsselresultate von Card und Moran zeigen jedoch, daß dies nicht der Fall ist.

Der trainierte Benutzer nimmt mit der Zeit eine Menge von wiederholten, einstudierten Verhaltensmustern an, von denen er annimmt, daß durch diese die Aufgabe am besten gelöst werden kann. (z.B. zurücktippen bei Fehlern in Textverarbeitung). Dabei ignoriert er sogar Funktionen des Programms, die er nicht in seinem Muster aufgenommen haben möchte.

Hat ein Benutzer beispielsweise zwei Buchstaben in einem Wort vertauscht, so möchte er diese wieder in die richtige Reihenfolge bringen. Es gibt Texteditoren wie den von UNIX bekannten „vi", die eine solche Funktion bieten. Arbeitet man mit verschiedenen Editoren, die nicht alle die gleiche Funktionalität besitzen, so wird man sich die Funktion „Buchstaben tauschen" nicht unbedingt merken. In Texteditoren ohne diese Funktion, wird man den Cursor an die entsprechende Stelle setzen, einen der beiden falschen Buchstaben löschen, den Cursor versetzen und den nun fehlenden Buchstaben wieder eingeben. Diese Vorgehensweise wird man irgendwann auch bei dem Texteditor benutzen, der eigentlich die andere Funktionalität bietet. Das erstaunliche dabei ist, daß man mit der „komplizierten Methode" meistens auch noch schneller ist, da man dies gewohnt ist, während man sonst erst einmal überlegen muß, wie das Kommando „Buchstaben tauschen" denn nun ausgeführt wird.

Um nun die Verhaltensmuster vorherzusagen, wurde das GOMS-Modell aufgestellt. GOMS steht für Goals-Operators-Methods-Selection. Während des Trainingsprozesses lernt der Benutzer, die Grundoperationen so zu kombinieren, daß daraus Methoden zum Lösen seiner Aufgabe werden.

Dabei wurde festgestellt, daß es im Prinzip genügt, die Anzahl der Tastaturanschläge, Mausbewegungen und Denkintervalle aufzuaddieren, die für jede Aufgabe benötigt werden. Dazu kommt die Zeit des Rechners zum Antworten, für die Bewegung der Hand von Tastatur zur Maus usw. Dies wurde auch das Keystroke-Level-Modell genannt.

Die eigentliche Aufgabe der Benutzerschnittstelle liegt nun darin, die Anzahl der Aktionen, die zur Lösung einer Aufgabe benötigt werden, möglichst gering zu halten und die einzelnen Aktionen möglichst schnell auszuführen. Um die Denkzeit des Benutzers gering zu halten, darf die (reine) Kapazität des Kurzzeitgedächtnisses nicht überlastet werden, d.h. zur Erfüllung einer Aufgabe sollten 3 bis 4, besser weniger Schritte genügen.

Eine gute Methode, das Kurzzeitgedächtnis zu entlasten, ist die Benutzung von Fenstern, wann immer dies möglich ist. In Fenstern ist stets Information enthalten, welche parallel zu anderen Fenstern ständig zugänglich ist. Der Benutzer muß lediglich das Fenster betrachten, es eventuell nach vorne holen, und schon hat er die gewünschte Information. Ein gutes Beispiel sind die Sliders (Schieber) der Fenster, die angeben, wo man sich in einem Dokument befindet und wieviel man von ihm im Fenster sehen kann.

Der wichtigste Faktor jedoch ist das Erhöhen der Vorhersehbarkeit und das Erniedrigen der Unentschlossenheit. Das Unsicherheitsprinzip sagt aus, daß die Zeit zum Treffen einer Auswahl geringer ist, wenn weniger Alternativen zur Verfügung stehen. Ein System sollte vor allem mit dem Benutzer kommunizieren und aus diesem Feedback entsprechende Folgerungen ziehen. Dadurch kann der Anfänger die Bedienung des Systems leichter erlernen, und durch die Vorhersehbarkeit wird das System komfortabel für den trainierten Benutzer.

In praxi werden nur solche Menüpunkte auswählbar gemacht, die vom aktuellen Status aus durchgeführt werden können. Ist beispielsweise noch kein Block markiert, so kann er z.B. nicht kopiert oder verschoben werden (siehe Abb. 5.12). Die entsprechenden Menüpunkte sind in schwacher Grauschrift dargestellt.

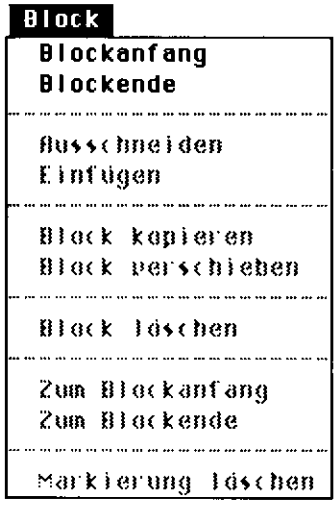



**Abbildung. 5.12:** „Block"-Menüpunkt in Wordplus

Dagegen ist der Menüpunkt „Blockanfang" anwählbar. Ist ein Blockanfang angewählt, so kommt jetzt der Menüpunkt „Zum Blockanfang" hinzu (Abb. 5.13).

**Abbildung 5.13:** „Block"-Menüpunkt mit definiertem Blockanfang

Wenn ein Menüpunkt ausgewählt wurde, sollte der Menütitel so lange invers (weiß auf schwarz) bleiben, bis die Operation komplett ausgeführt wurde. Dem Benutzer wird so angezeigt, daß er das richtige Menü angewählt hat. In unserem Beispielprogramm wird z.B. bei der Auswahl des Menüs Info über eine Funktionstaste diese invertiert bleiben, bis das Menü zu Ende geführt ist (s. Abb. 5.14). Dasselbe gilt für das Menü, aus dem die Funktionstaste entnommen wurde. Entsprechendes gilt auch für die anderen Tastenkombinationen, die zum Ausführen von Menüpunkten gedrückt werden müssen.

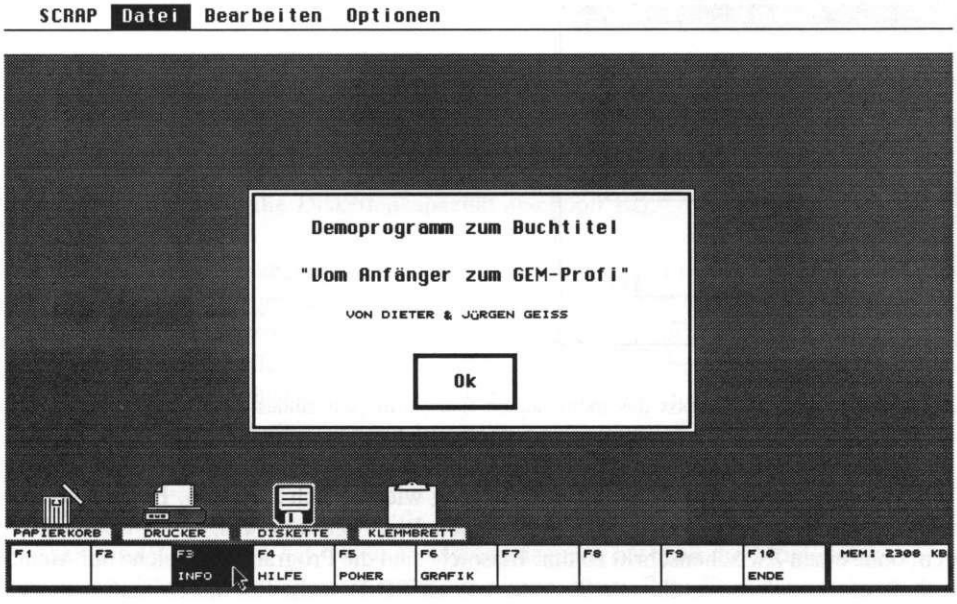



**Abbildung 5.14:** Funktionstaste und Menü aktiviert

Wird in unserem Beispielprogramm z.B. der Papierkorb geöffnet, so erscheint zunächst eine Vergrößerungsbox, die vom angewählten Objekt ausgeht und anzeigt, daß das entsprechende Objekt geöffnet wird. Das gibt dem Benutzer während des Ausführens der Operation die Sicherheit, das richtige Objekt benutzt zu haben. Beim Schließen des Fensters wird entprechend die Verkleinerungsbox gezeichnet, so daß nachträglich noch festgestellt werden kann, aus welchem Objekt das Fenster geöffnet wurde. Das Zeichnen dieser Boxen kostet aber eine bestimmte Zeit, so daß es günstig für den trainierten Benutzer ist, wenn er diese optische Anzeige ausschalten kann. Ein Beispiel hierfür ist Tempus 2.0.

Während der Rechner mit einigen Operationen beschäftigt ist, zeigt er ein Stundenglas (MS-DOS, FlexOS) oder eine Biene (ATARI ST). Bei länger dauernden Operationen sollte der momentane Stand der Bearbeitung ständig ausgegeben werden, so daß der Benutzer die Arbeit des Rechners regelrecht verfolgen kann. Beispiele sind die Ausgabe aller Datensätze auf Drucker oder Diskette im Datenbanksystem ADIMENS ST (Abb. 5.15). Dort erfährt der Anwender, welche Operation gerade läuft, wie sie abgebrochen werden kann, wie lange sie bereits läuft und wie lange sie noch läuft. Die digitale Anzeige in Sekundenbruchteilen muß allerdings erst vom Anwender dekodiert werden. Schneller aufzufassen ist die analoge Anzeige, die hier durch einen Balken dargestellt wird, der nach rechts wandert. Hier erkennt der Anwender sofort, wie weit die Operation fortgeschritten ist. Die Anzeige ist zwar ungenauer als eine digitale, aber meist genügt sie. Der Anwender darf nie im Unklaren gelassen werden, welche Operation gerade läuft und wie weit sie fortgeschritten ist!

**Abbildung 5.15:** Anzeigebox des momentanen Bearbeitungszustandes

Wichtig ist auch, die Anzahl der Modi so gering wie möglich zu halten. Ein Modus existiert genau dann, wenn nicht alle Möglichkeiten des Programms ausgenutzt werden können, ohne einen Zwischenschritt zu tun. Beispiele sind die Programme, welche nur Menütechnik anwenden (hierarchische Systeme wie UCSD-Pascal). Bei diesen muß man sich durch einen Dschungel von Menüs und Untermenüs durchhangeln, bis man sich an der

gewünschten Stelle befindet. Durch diese Hierarchie muß man sich dann zuerst wieder nach oben hangeln, wenn man auf einen anderen Zeig wechseln möchte.

Beim Lösen einer Aufgabe sieht der Benutzer nun für jeden Modus einen zusätzlichen Zustand in seinem Problemraum. Diese zusätzlichen Zustände müssen analysiert werden, was Denkzeit kostet. Eine Benutzerschnittstelle ohne Modi nennt man auch transparent, weil die Erfüllung von Aufgaben nicht an bestimmte Zustände geknüpft ist.

Ein Beispiel ist das Ausdrucken einer Liste aller Datensätze in ADIMENS ST. In manchen Programmen muß man zuerst ein Menü anwählen, um das Ausgabegerät auf den Drucker umzustellen. Danach kehrt man zurück, um in einem anderen Menü die Datensätze anzuwählen und auszugeben. In ADIMENS ST wird einfach das Symbol des Karteikastens auf das Symbol des Druckers gelegt. Damit beginnt sofort die Ausgabe. Sollen die Datensätze auf Diskette ausgegeben werden, kann man ebenfalls ein entsprechendes Symbol (die Diskstation) benutzen.

Das Benutzen von Fenstern ist ebenfalls eine Methode, Modi zu eliminieren. Z.B. ist es im Texteditor TEMPUS möglich, Dateien gleichzeitig in mehreren Fenstern zu bearbeiten, ohne daß ein Zustand eingeschaltet oder verlassen werden muß. In herkömmlichen, älteren textorientierten Systemen kann z.B. immer nur eine Datei bearbeitet werden. Ähnliches gilt für das Programm Signum. Man hat immer nur die Möglichkeit, ein Dokument zu einer Zeit zu verarbeiten. Möchte man eben mal in einem anderen Dokument etwas ändern, so muß man das aktuelle Dokument abspeichern, das neue Dokument laden, die Änderung vornehmen, das Dokument abspeichern, das alte Dokument laden und schließlich sogar noch die Stelle suchen, an der man sich das letzte Mal befunden hat. Durch diese Vielzahl von Operationen wird das Kurzzeitgedächtnis erheblich belastet, man bekommt ein ungutes Gefühl.

Ein Modus besteht bei GEM immer, wenn eine Dialogbox oder eine Fehlermeldung auf dem Bildschirm erscheint. Bei Fehlermeldungen erscheint dieser Umstand noch tragbar, da der Benutzer den Fehler zur Kenntnis nehmen muß und das System eventuell nach Auftreten des Fehlers nicht mehr so reagiert, wie der Benutzer dies wollte (z.B. Fehler beim Abspeichern, da die Diskettenkapazität erschöpft ist).

Bei einer Dialogbox hingegen muß diese erst positiv (OK) oder negativ (ABBRUCH) verlassen werden, um den gewünschten Effekt zu erwirken. Dies bedeutet, daß in der Zeit, in welcher eine Dialogbox abgearbeitet wird, keine andere Aktion stattfinden kann. Insbesondere können keine Menüs angewählt werden. Versuche haben gezeigt, daß Personen, die sich nicht im GEM auskannten, diesem Umstand keine Rechnung trugen und verzweifelt versuchten, Menüs anzuwählen. Tatsächlich muß es für einen ungeübten Benutzer merkwürdig sein, auf einmal keine Menüs mehr auswählen zu dürfen.

Aus diesem Dilemma gibt es im Prinzip keinen Ausweg, da der Dialogmanager des GEM dies genauso vorgesehen hat. Beim Macintosh dagegen hat man die Möglichkeit, sogenannte „modeless dialog boxes" zu programmieren. Eine solche Dialogbox befindet sich in einem Fenster. Dadurch können weiterhin Menüs und andere Boxen geöffnet werden,

ohne daß die aktuelle Dialogbox verlassen wird. Es gibt natürlich auch die Möglichkeit, solche Boxen in GEM zu programmieren. Nur hat man hier einen erheblich größeren Aufwand. Wir haben diese Möglichkeit zunächst noch offen gelassen, jedoch schon vorgesehen. Möglicherweise wird in einer der nächsten Auflagen ein Dialogmanager ohne Modus in das Beispielprogramm eingebunden sein.

Wichtig ist dies auch für die gleichzeitige Verwendung mehrerer Programme, wie dies zum Beispiel bei X/GEM der Fall ist. Die Verwendung von Dialogboxen ist dort tatsächlich ein Problem, weil innerhalb der Box nicht auf eine andere Applikation umgeschaltet werden kann. Das Umschalten geschieht bekanntlich durch Anklicken eines zur Anwendung gehörenden Fensters oder durch Auswählen der Applikation aus dem aktiven Menü des obersten Fensters. Benutzung von Dialogboxen ohne Modi unterstützt somit die Multi-Tasking-Eigenschaften eines Systems.

Eine andere Definition besagt, daß Programme transparent sind, wenn jede Aktion genau ein Resultat zur Folge hat. Dieser Umstand sollte möglichst immer beachtet werden. In Dialogboxen wird dies automatisch durch den Dialogmanager des GEM eingehalten. Ähnliches gilt für Fenster. Es hat allerdings Programme wie eine frühere Version des SPC-Modula gegeben, bei denen die Bedeutung der Schließbox und der Vergrößerungsbox vertauscht waren! Der Benutzer kann das Resultat in diesem Fall nicht mehr vorhersehen. Ähnliches gilt für die Programme Signum und STAD. Beide benutzen die linke und rechte Maustaste. Möchte man bei Signum um eine Seite vorwärtsblättern, benutzt man die rechte Maustaste, beim Rückwärtsblättern die linke. Bei STAD ist es genau umgekehrt! Möchte man die nächste Bildschirmseite sehen, so klickt man mit der linken Taste in das entsprechende Feld, für die vorige Seite benötigt man die rechte Taste.

Die Resultate, die von Aktionen erzeugt werden, sollten bei möglichst allen Programmen, die auf einem Rechner laufen, die gleichen sein. Bei Rechnern wie dem Macintosh wird dies hervorragend eingehalten, so daß ein Anwender, der ein Programm bereits bedienen kann, sich sehr schnell in anderen Programmen zurechtfindet.

Das einzige GEM-Programm, das jeder nach kurzer Zeit bedienen kann, ist der GEM-Desktop, da er beim Einschalten des Rechners als erste Applikation zur Verfügung steht und ständig benutzt wird. Programme, die sich so verhalten wie der GEM-Desktop, wird jeder ATARI ST-Benutzer schnell bedienen können.

Bei ADIMENS ST oder dem Textdeditor TEMPUS wurde auf eben genannte Tatsache großen Wert gelegt. In TEMPUS wird z.B. der Text gelöscht, wenn das entsprechende Piktogramm auf den Papierkorb gelegt wird.

In ADIMENS ST kann das Löschen eines Datensatzes genauso geschehen wie das Löschen einer Datei auf dem GEM-Desktop, d.h. ein Fenster öffnen, um den Inhalt anzuzeigen und dann den entsprechenden Datensatz oder die ganze Datei in den Papierkorb legen.

Auch das Selektieren von mehreren Objekten in einem Bildschirmfenster sollte immer in gleicher Weise vonstatten gehen. Eine Möglichkeit ist das Einrahmen (dragging) von

Objekten mit Hilfe des Gummibandes (dragbox). Alle Objekte, die das Gummiband schneiden, werden selektiert. Eine andere Möglichkeit ist die Auswahl mit Hilfe der Shift-Taste und dem Mausklick. Damit können Objekte, die nicht zusammenstehen, ausgewählt werden.

Eine neue Methode, mehrere Zeilen in einem Texteditor auszuwählen, ist das Benutzen eines unsichtbaren Gummibandes, wobei während des Aufziehens alle Zeilen, die überstrichen werden, selektiert werden (siehe Abb. 5.16). Dies wird beispielsweise im Editor des Turbo-C-Entwicklungspakets benutzt. Der größte Vorteil dieser Methode ist es, daß der Benutzer die Möglichkeit hat, die Reaktion auf seine Aktion in Echtzeit zu verfolgen und Fehler sofort korrigieren zu können, noch bevor die Aktion abgeschlossen ist. Im Prinzip wird hier ein weiterer Modus abgeschafft, da das Resultat nach Loslassen der Maustaste sofort so zur Verfügung steht, wie es angezeigt wurde (What you see is what you get).

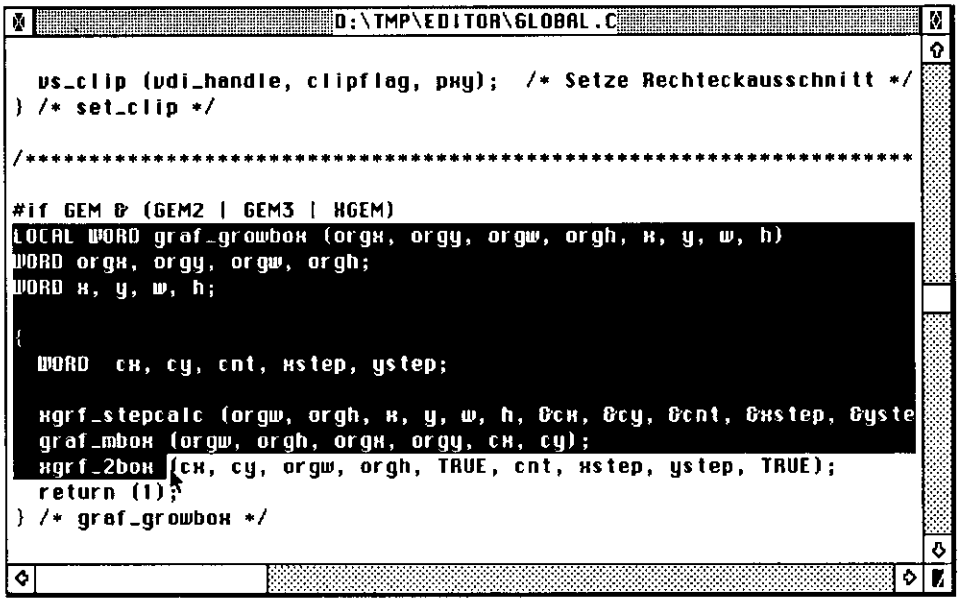



**Abbildung 5.16:** Selektieren von mehreren Zeilen in Echtzeit

Eine bekannte Methode beim Ziehen von Objekten ist auch das automatische Invertieren der überstrichenen Objekte. Beim GEM-Desktop ist dies bekannt, wenn z.B. eine Datei gelöscht oder kopiert werden soll. Noch während des Ziehens wird das überstrichene Objekt invertiert, um anzuzeigen, daß beim Loslassen des Mausknopfes diese Aktion ausgeführt wird. Leider gibt es dafür keine Routine in GEM, die dies automatisch erledigt. In unserem Beispielprogramm ist allerdings eine solche Routine enthalten. Sie ist allgemein geschrieben und kann für beliebige Objekte benutzt werden (siehe auch Abb. 5.17).

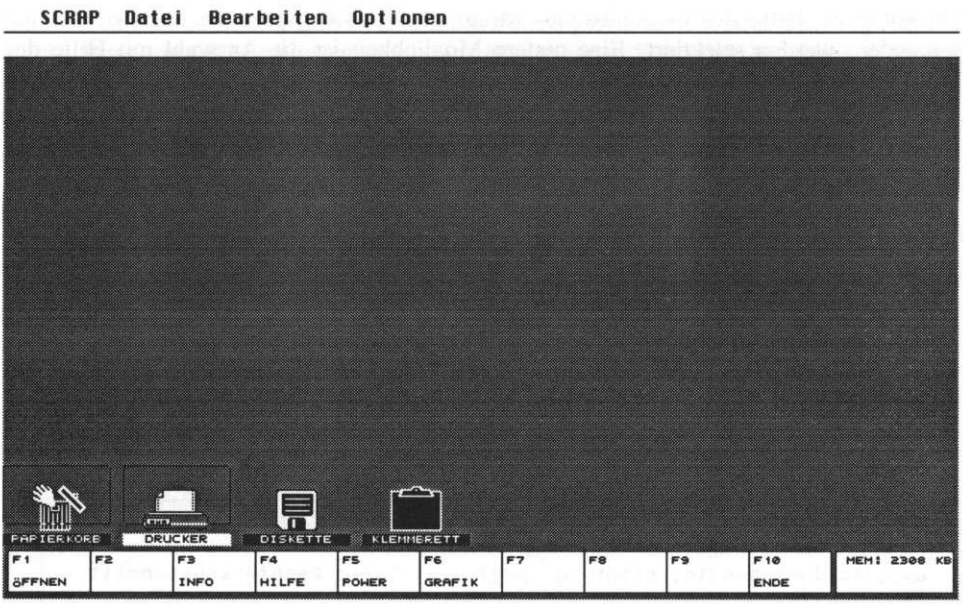



**Abbildung 5.17:** Ziehen von Objekten mit Invertieren des Ziels

### 5.2.6 Übersichtlichkeit von Dialogboxen

Um Dialogboxen übersichtlich zu gestalten, hat man eine Menge von Möglichkeiten. Wichtig sind folgende Grundprinzipien:

- Nicht zu viele Objekte auf einer Ebene verwenden.
- Klare Abgrenzung zwischen Anzeigen und aktivierbaren Schaltern
- Gleicher Aufbau aller Dialogboxen einer Applikation
- Möglichst gleicher Aufbau der Dialogboxen aller Applikationen
- Benutzung der gleichen Buttons mit gleicher Wirkung
- Untergliederung der Information in Blöcke
- Jede Dialogbox sollte eine abgeschlossene Teilaufgabe behandeln
- Überflüssige Informationen weglassen
- Numerische Daten digital anzeigen, wenn genaues Ablesen erforderlich
- Numerische Daten analog anzeigen, wenn Schätzen angebracht ist
- Eventuell Daten sowohl digital als auch analog anzeigen
- Blinken, Inversdarstellung und Farben sparsam benützen

Um einige der oben angesprochenen Punkte zu veranschaulichen, betrachten wir die folgende Dialogbox „Druckereinstellung“ zur Steuerung einer Druckausgabe (Abb. 5.18).

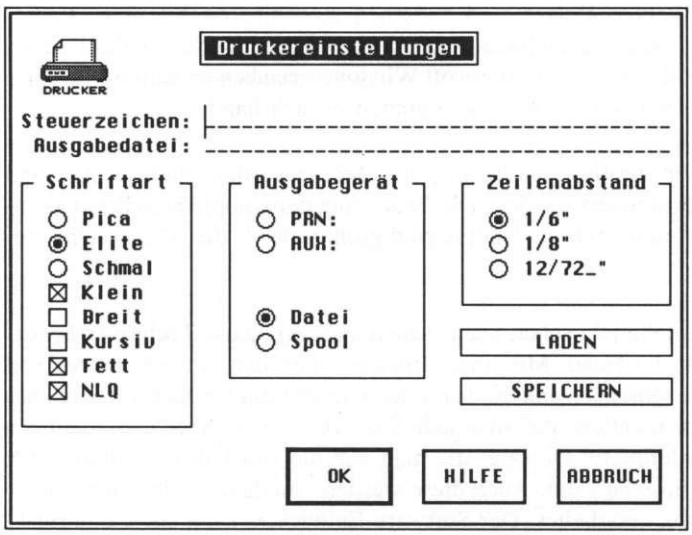



**Abbildung 5.18:** Beispielhafte Dialogbox

Die oben dargestellte Dialogbox erfüllt die oben angegebenen Forderungen in folgender Weise:

Von oben nach unten betrachtet gibt es zunächst einmal den Titel der Dialogbox, der anzeigt, wo man sich überhaupt befindet. Links daneben befindet sich das Icon, welches angeklickt wurde, bevor sich die Dialogbox öffnete. Das Symbol des Druckers ist schneller vom visuellen System zu verarbeiten als die Überschrift.

Nach zwei editierbaren Zeilen, von denen die zweite nur dann zu editieren ist, wenn das Ausgabegerät auf Datei umgestellt ist, folgen drei Blöcke von Informationen. Die Blockbildung wird hier auf dieselbe Weise vorgenommen, wie dies in Macintosh-Programmen üblich ist. Es gibt einen Rahmen, der einen Block begrenzt. Dieser hat eine Überschrift, die sich am oberen Blockrand befindet, aber noch zu diesem gehört.

Diese Art der Überschrift ist in GEM nicht ohne einen kleinen Trick möglich, da bei verschiedenen Auflösungen die halbe Schrifthöhe nicht immer eine feste Anzahl von Pixeln beträgt. Eine solche Blocküberschrift kann in einem Resource-Construction-Set durch Hilfe des erweiterten Typs (siehe auch Kapitel 2) definiert werden. In dem Modul „Resource" unseres Beispielprogramms wird die Überschrift automatisch richtig plaziert.

In einem solchen Block befinden sich die Einstellungen, wobei unterschieden werden muß zwischen normalen Knöpfen und Radio-Knöpfen. Bei Radio-Knöpfen kann immer nur genau eine Alternative aktiv sein. Beim Aktivieren einer Alternative erlischt entsprechend eine andere Alternative. Normale ankreuzbare Knöpfe können ein- und ausgeschaltet werden.

GEM selbst macht nun keinen Unterschied zwischen anwählbaren Knöpfen, Radio-Knöpfen und Aktions-Knöpfen, die auch eine Aktion wie z.B. das Verlassen des Dialogs verursachen. Der Macintosh, aber auch Microsoft Windows erlauben es, durch einen einzigen Blick zu erkennen, um welche Art von Knöpfen es sich handelt:

Anwählbare Knöpfe sind quadratisch und können angekreuzt werden. Radio-Köpfe sind rund und können ebenfalls aktiviert werden. Die Bedeutung der Knöpfe befindet sich immer rechts neben den Knöpfen. Aktions-Knöpfe sind größer, da sie die Aktion selbst beherbergen.

Wie man am obigen Beispiel leicht sehen kann, haben auch wir diese Techniken bevorzugt. Der Vorteil liegt auf der Hand. Mit einem Blick erkennt man, um welche Art von Knopf es sich handelt. Die fehlenden ankreuzbaren Knöpfe und runden Radio-Knöpfe haben wir so allgemein implementiert, daß man jede Dialogbox damit ausstatten kann. Es handelt sich um benutzerdefinierte Objekte, die man wie die Blocküberschriften durch einen erweiterten Typ definieren kann. Auch diese werden von dem Modul „Resource" unseres Beispielprogramms verarbeitet. Der Software-Entwickler kann sie wie normale Buttons behandeln. Damit kommt man dem Macintosh-Standard bei Dialogboxen schon sehr nahe.

Die Aktions-Knöpfe in dieser Dialogbox lauten OK, HILFE und ABBRUCH. Es sind Standard-Knöpfe, die immer angeboten werden sollten. Sie bewirken eine bestimmte Ebene von Konsistenz innerhalb eines Programms aber auch zwischen den Programmen. OK bewirkt ein Verlassen der Dialogbox, wobei die Einstellungen übernommen werden. Entsprechendes gilt für ABBRUCH, wobei die Einstellungen aber nicht übernommen werden sollen. Bei Betätigen des HILFE-Knopfes wird die Dialogbox kurzzeitig verlassen. Es erscheint eine Hilfe-Tafel, die dem Benutzer zeigt, wie er die Dialogbox zu verstehen hat. Das Thema Hilfe wird im nächsten Abschnitt noch einmal ausführlicher aufgegriffen.

Wir haben bereits oben gesehen, wie man mit Hilfe von Pop-Up-Menüs die Mauswege optimieren kann. Pop-Up-Menüs haben aber noch eine andere Bedeutung. Sie können auch in Dialogboxen günstig eingesetzt werden, um etwa Information zu verstecken, die dann auf Mausklick angezeigt wird. Der tiefere Sinn liegt wieder in der begrenzten Kapazität des Kurzzeitgedächtnisses. Bei zu vielen Informationen gleichzeitig wird dieses total überlastet.

Die Abbildung 5.10 (siehe oben) zeigt eine einfache Dialogbox, bei der die Eingabe des Feldes „Anrede" einer Karteikarte über ein Pop-Up-Menü geregelt wird.

Um nun zu erkennen, welche Felder versteckte Informationen beinhalten, sollten wegen der Vorhersehbarkeit, die dem Benutzer geboten werden sollte, ein eindeutiges und einheitliches Aussehen erreicht werden. Seit dem Frühjahr 1988 sind Pop-Up-Menüs im Macintosh-Betriebssystem implementiert. Wir schlagen vor, das Aussehen an das vom Macintosh anzugleichen. Das bedeutet, daß Pop-Up-Menüs in Dialogboxen einfache Rahmen mit Schatten haben. Entsprechend sollten die Felder, die Pop-Up-Menüs beher-

bergen, eingerahmt sein und ebenfalls einen Schatten zeigen. Sie sehen durch diese pseudo-dreidimensionale Darstellung aus, als könnten sie noch Information in der dritten Dimension (der Höhe) verbergen.

Die Benutzung eines Pop-Up-Menüs wurde schon weiter oben erläutert.

Ein letztes Wort zu den Dialogboxen. Viele Entwickler erweitern während der Programmierung den Inhalt der Dialogboxen. Doch viele Entwickler können sich nicht durchringen, die Dialogbox danach zu sortieren. Die meisten Menschen lesen von oben nach unten bzw. von links nach rechts. Genauso sollten auch die Inhalte von Dialogboxen sortiert sein. Beim Aufbau der Box ergibt sich dann ein gleichmäßiges Bild.

Werden sie nicht sortiert, so erscheinen die einzelnen Elemente wild durcheinander, unten, oben, rechts, dann wieder in der Mitte usw. Dies verwirrt den Benutzer, da er seine Aufmerksamkeit auf die Reihenfolge des Aufbaus legt. Außerdem kann er nicht von oben nach unten lesen, während die Box sich schon aufbaut, da am Ende noch Information zugänglich gemacht werden könnte, die zuerst fehlt. Beispiele solcher unsortierter Dialogboxen findet man in Programmen wie Signum oder Turbo C.

Das gleiche gilt übrigens auch für Menüs. Sie sollten sich immer von oben nach unten aufbauen. Bestimmte Resource-Construction-Sets übernehmen glücklicherweise die Sortierung automatisch, so daß man sich nicht selbst darum kümmern muß.

## 5.3 Fehlerbehandlung

Ein wichtiges Kapitel beim Aufbau von Benutzeroberflächen ist die Fehlerbehandlung und die Hilfestellung des Programms. Bei der Fehlerbehandlung unterscheidet man Fehlervermeidung und Fehleraufhebung.

Bei der Fehlervermeidung wird vom System darauf geachtet, daß der Benutzer keinen Fehler machen kann. Dazu gehören z.B. die abgeschalteten (grauen) Menüpunkte, die ein Benutzer nicht anwählen kann, wenn nicht ein bestimmter Zustand vorliegt. Durch das Verhindern der Menüauswahl kann der Benutzer in diesem Sinne zumindest kein Menü anwählen, welches, wäre es anwählbar, einen Fehler erzeugen würde.

Ähnliches gilt für die Optionen einer Dialogbox, welche grau dargestellt sind, wenn sie nicht angewählt werden können.

Ein optimales System wäre eines, bei dem nie ein Fehler auftreten kann. Dies ist jedoch praktisch nicht zu verwirklichen – zumindest nicht softwaremäßig. So kann man beim Atari oder beim IBM PC jederzeit die Diskette im Laufwerk wechseln, auch während Schreiboperationen durchgeführt werden. Beim Macintosh ist dies z.B. nicht möglich, hier kontrolliert die Software den Diskettenauswurf. Fehler durch Unachtsamkeit beim Diskettenwechsel können erst gar nicht auftreten.

Fehleraufhebung bedeutet ein Rückgängigmachen einer Operation. Man nennt dies auch eine „Undo"-Funktion. Optimal wäre die Realisierung eines Systems, bei dem man zu jedem Zeitpunkt jede frühere Situation wieder herstellen kann, d.h. ein Rückgängigmachen von beliebig vielen Schritten. Dies ist jedoch ohne enormen Aufwand nicht so einfach möglich. Beispiele sind Datenbanken mit Transaktionsmechanismen oder das SCCS-System unter UNIX, bei dem nur die Differenzen von Quelltexten gespeichert werden. Man kann dann jederzeit bestimmen, welche (zeitliche) Version eines Quelltextes man zur Verfügung haben möchte. Dazu gehört allerdings eine gewisse Disziplin beim Schreiben der Quelltexte.

Viele Programme, vor allem Texteditoren bieten inzwischen Undo-Funktionen, um die letzte Aktion rückgängig zu machen. Dies gilt jedoch nicht für alle Operationen. Ein globales Suchen und Ersetzen kann nur von den wenigsten Texteditoren wieder rückgängig gemacht werden. Bei den meisten Grafikprogrammen (z.B. STAD) gibt es ebenfalls Undo-Möglichkeiten.

Der GEM-Desktop ist auch nicht gerade ein Paradebeispiel für Undo-Funktionalität. Eine gelöschte Datei ist unwiderruflich verloren. Besser hat man es auf DOS-Rechnern, wo man gelöschte Dateien einigermaßen bequem mit Hilfe von Utilities zurückholen kann. Die beste Lösung allerdings bietet der Finder des Macintosh, bei dem man gelöschte Dateien einfach wieder aus dem Papierkorb zieht.

Eine Undo-Funktion für Dialogboxen bietet immerhin der ABBRUCH-Knopf. Bei Fenstern können auch bestimmte Funktionen leicht rückgängig gemacht werden. Ein Scrolling nach unten kann durch ein Scrolling nach oben leicht durch den Benutzer kompensiert werden. Beim Schließen eines Fensters sollte darauf geachtet werden, daß, wenn sich das Fenster in ein Icon „verwandeln" kann, durch ein erneutes Öffnen des Icons das Fenster sich wieder an der alten Position befindet. Hervorragend gelöst ist dies z.B. beim Texteditor TEMPUS.

Das Undo bietet dem Benutzer die nötige Sicherheit, mit dem System zunächst zu experimentieren, ohne daß er Sorgen haben müßte, daß seine Arbeitsergebnisse durch unsachgemäße Bedienung verloren sind. Wir können hier nicht genug darauf hinweisen, daß möglichst jedes Programm eine Undo-Funktion haben sollte. Als Bedienung bietet sich die Undo-Taste an. Da diese jedoch beim IBM PC nicht existiert, sollte man hier auf eine Funktionstaste oder eine Control-Kombination ausweichen. Control-Z hat sich hier bereits als Standard durchgesetzt.

## 5.4 Hilfe

Ein ausgeklügeltes Hilfesystem ist enorm wichtig. Es entlastet das Langzeitgedächtnis und bietet zu jedem Zeitpunkt eine angemessene Hilfe. Ein gutes Hilfesystem bietet das Entwicklungspaket von Turbo-C. Zu jedem Zeitpunkt kann ein Stichwortverzeichnis aufgerufen werden, wobei die Stichworte selbst fett gedruckt sind. Bei Anklicken eines Stichwortes verzweigt das Programm zum entsprechenden Kapitel. Dort befinden sich

meist Querverweise, die wiederum fett gedruckt sind und als Einstiegspunkt in ein anderes Kapitel dienen. Auf diese Weise spart man sich das ständige Suchen im Handbuch.

Für Programmierer ist ein solches System ideal, da es eine große Menge von Funktionen gibt, deren Parameter er sich nicht immer merken kann. Ein Anwender hat aber andere Vorstellungen von einer Hilfestellung. Er möchte zu jedem Zeitpunkt eine Hilfestellung aufrufen können, die sich auf die aktuelle Situation bezieht. Dies nennt man auch eine kontextsensitive Hilfe.

Bereits 1982 hat Nievergelt die vier wichtigen Fragen formuliert, auf die ein Dialogsystem jederzeit dem Benutzer Auskunft geben muß:

- Wo bin ich?
- Was kann ich hier tun?
- Wie kam ich hierhin?
- Wo kann ich hin, und wie komme ich dorthin?

In eine Applikation sollte nun eine Hilfestellung derart eingebaut werden, daß bei Betätigen des Menüpunktes „Hilfe" oder Drücken der Taste „Help" bzw. Anwählen des Knopfes „Hilfe" die oben erwähnte Hilfestellung erscheint. Die Hilfemeldungen sollten dann kontextabhängig gegeben werden. Das bedeutet im einzelnen:

a) In jeder Dialogbox muß sich ein Hilfe-Knopf befinden, bei dessen Betätigung eine Hilfemeldung über gerade diese Dialogbox ausgegeben wird.

b) In jeder Fehlermeldung muß sich ein Hilfe-Knopf befinden, bei dessen Betätigung eine entsprechende Hilfemeldung über diesen Fehler ausgegeben wird. Es muß dem Benutzer klargemacht werden, warum der Fehler aufgetreten ist und wie er zu beheben ist.

c) Ist ein Fenster offen und keine Dialogbox oder Fehlermeldung auf dem Bildschirm, so muß zu diesem Fenster eine entsprechende Hilfemeldung ausgegeben werden.

d) Ist kein Fenster und keine Dialogbox offen (der normale Desktop ist dann sichtbar), so kann ein genereller Hilfstext, welcher sich auf das Programm selbst bezieht, ausgegeben werden.

Bei Betätigen des Hilfe-Knopfes in Dialog- und Alertboxen erscheint die Hilfemeldung mit folgenden Informationen:

- Wo bin ich?

In einer Dialogbox sollte am oberen Rand immer der Titel dieser Box stehen. Dieser sollte irgendwie hervorgehoben werden (invertiert, schattiert etc.) und für alle Dialogboxen der Applikation gleich aussehen. Diese Regel gilt auch für Hilfe-Dialogboxen.

– Was kann ich hier tun?

Für jeden Knopf, der anwählbar ist oder eine Aktion auslöst, sollte eine Erklärung in der Hilfe-Dialogbox gegeben werden.

– Wie kam ich hierhin?

Die Antwort sollte dem Benutzer aufzeigen, welchen Weg er eingeschlagen, beziehungsweise welche Aktionen er bis dahin ausgelöst hatte. Dies gilt besonders für Hilfe bei Fehlermeldungen.

– Wo kann ich hin, und wie komme ich dorthin?

Bei den Standard-Dialogboxen, ist die Antwort schon vorgegeben. Der OK-Knopf bewirkt ja das positive, der Abbruch-Knopf das negative Verlassen der Box. Der Hilfe-Knopf zeigt den Hilfetext an. Existieren weitere Knöpfe, die eine weitere Dialogbox öffnen, wie zum Beispiel das Laden einer Einstellung, so muß dies erklärt werden.

Für die Möglichkeit, Hilfetexte auch aus Alertboxen heraus auszugeben, wurde eine Routine implementiert, der die Fehlermeldung und ein Objektbaum für den Hilfetext mitgegeben wird. Wird nun in der Fehlermeldung der Hilfe-Knopf gedrückt, so wird dieser Objektbaum auf dem Bildschirm ausgegeben. Der Hintergrund wird wie bei einer Alertbox gerettet, so daß der ursprüngliche Kontext nach Verlassen der Hilfsbox nicht verlorengeht. Abbildung 5.19 zeigt eine Fehlermeldung, Abbildung 5.20 die aus dem Anwählen des Hilfekopfes resultierende Dialogbox.

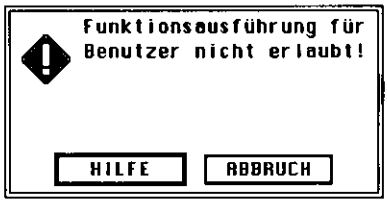



**Abbildung 5.19:** Beispiel-Fehlermeldung

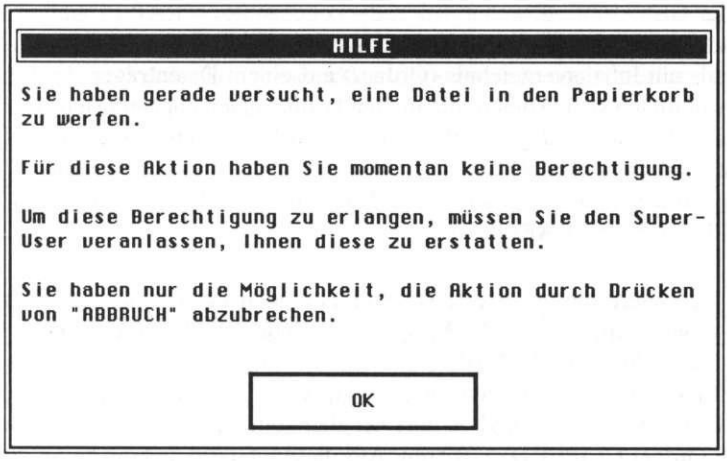



**Abbildung 5.20:** Hilfe-Dialogbox zu obiger Fehlermeldung

Am besten wäre natürlich ein Hilfesystem in einem echten Fenster, welches man parallel zum eigentlichen Arbeitsgang geöffnet haben kann. Konsequenterweise müßte man bei einem solchen System aber auf Dialogboxen verzichten, da aus diesen kein Fenster geöffnet werden kann. Erst die „modeless dialog boxes" (s.o.) würden ein solches System möglich machen. Doch diese existieren noch nicht für GEM.

Es muß hier leider auch der Aspekt der Raubkopien kurz angesprochen werden. Je besser ein Hilfesystem, desto leichter kann eine Raubkopie ohne ein dickes Handbuch weitergegeben werden. Aus diesem Grund muß jeder Entwickler selbst entscheiden, wie gut er sein Hilfesystem implementieren möchte.

## 5.5 Datenaustausch

Ein wichtiges Kapitel stellt auch der Datenaustausch dar. Bei Rechnern wie dem Macintosh hält sich jedes Programm an die vorgegebenen Standards, so daß mit Leichtigkeit Daten von einem Programm in ein anderes übernommen werden können.

Aus irgendeinem Grund waren aber Entwickler von GEM-Software nicht in der Lage, sich aufzuraffen und ebenfalls Standards zu folgen. Das Resultat davon sind zwar oft recht gute Programme, aber fast alle Programme haben ihre eigenen Formate für Texte und Grafiken. Dies ist vor allem deswegen unverständlich, weil es durchaus Richtlinien gibt, die von Digital Research beschrieben wurden, an die sich aber niemand gehalten hat.

Doch nun beginnen die Software-Entwickler umzudenken. Allen voran haben die Entwickler von 1st Wordplus in ihrer neuesten Version den Datenaustausch möglich gemacht. Wie geht dies nun vor sich?

In der Beschreibung der GEM-Bibliotheken existiert die sogenannte SCRAP-Bibliothek. In ihr befinden sich Aufrufe zur Verwaltung eines Klemmbretts (Clipboard). Das Klemmbrett ist nichts weiter als ein Inhaltsverzeichnis (Ordner) auf einem Datenträger. In diesem Inhaltsverzeichnis befinden sich Dateien, die in Standardformaten abgelegt sind. Der Name des Verzeichnisses hat sich in den verschiedenen GEM-Versionen gewandelt. Bis zur Version GEM 2.X hieß dieses GEMSCRAP und befand sich – zumindest beim PC – auf dem Laufwerk, auf dem sich das gesamte GEM-System (GEMBOOT, GEMSYS, GEMDESK, GEMAPPS, GEMSCRAP) befand.

Ab GEM/3 hat sich dies allerdings geändert. Da hier nur noch ein Ordner, nämlich GEMAPPS existiert, in welchem sich das GEM-System befindet, muß sich der Ordner in diesem befinden. Er heißt hier CLIPBRD, der vollständige Zugriffspfad mithin C:\GEMAPPS\CLIPBRD\, falls sich das System auf Laufwerk C befindet. Da das neue Wordplus auch für GEM/3 geeignet ist und es dort Möglichkeiten zum Datenaustausch gibt, muß ebenfalls der Ordner CLIPBRD existieren. Aus diesem Grund und um kompatibel zu späteren Versionen zu bleiben, schlagen wir diesen Namen vor.

Bei X/GEM unter FlexOS hat sich der Ordner C:\BIN\SCRAP\ eingebürgert.

Innerhalb des Ordners können sich nun diese Dateien befinden. Das Inhaltsverzeichnis ist nur eine Ebene tief. Das bedeutet, daß zwei Operationen, die jeweils etwas in diesem Verzeichnis speichern, die letzte Version überschreiben. Jedes Programm sollte nun die Möglichkeit bieten, auf diese Dateien mittels folgender Operationen zuzugreifen:

Ausschneiden (Cut): Der vorher ausgewählte Block wird ausgeschnitten und aus dem Original entfernt. Vorher wird er auf das Clipboard geschrieben.

Kopieren (Copy): Wie „Ausschneiden", aber ohne Löschen des Originals.

Einfügen (Paste): Aus dem Clipboard wird das entsprechende Stück gelesen und an der aktuellen Cursorposition eingefügt.

Da das Lesen und Schreiben auf Diskette oder Platte im Vergleich zum Hauptspeicher recht langsam geht, kann man die Cut/Copy/Paste-Funktionen so implementieren, daß es davon jeweils zwei gibt. Eine Funktion arbeitet nur im Speicher, die andere auf dem Clipboard. In Wordplus ist dies auf die eben beschriebene Weise realisiert. Das Clipboard heißt dort Puffer. Zusätzlich gibt es noch die Möglichkeit, an den Puffer im Clipboard etwas anzuhängen. Es bietet sich also auch noch die Funktion Anhängen (Append) an. Das Anhängen ist einfach für Textdateien zu implementieren, wird jedoch für Grafik schwierig, wenn nicht der gesamte Inhalt vorher eingelesen werden soll.

Um Menüs zu sparen, sind in unserer Beispielapplikation die jeweiligen Menüpunkte für das Bearbeiten nur einmal vorhanden. Ein Schalter regelt dann die Ausführung der Operationen derart, daß sie entweder im Speicher oder auf dem Clipboard (auf den GEM-Klemmbrett) geschehen (s.o.).

Die Standardformate sollten von jedem Programm verarbeitet werden. Digital Research selbst hat bereits Standardformate vorgegeben, und unser Vorschlag ist es, sich genau an diese zu halten. Die Standardformate werden in den Unterlagen zu GEM 2.X, GEM/3 und X/GEM angegeben.

Alle Dateien im Clipboard, auch Scrap genannt, haben den gleichen Namen, aber jeweils einen verschiedenen Suffix. Folgende Typen bzw. Dateibezeichnungen werden vorgeschlagen:

SCRAP.CSV: Die Abkürzung bedeutet Comma-Separated-Values. Leider befindet sich in der Originaldokumentation keine genaue Beschreibung dieses Formats. Wahrscheinlich handelt es sich einfach um eine „endlose" Liste von Zahlen oder Zeichenketten, die jeweils durch Kommata getrennt sind.

SCRAP.TXT: Normaler ASCII-Text ohne spezielle Zeichen. Alle druckbaren Zeichen des Zeichensatzes auf dem jeweiligen Rechner können vorkommen. Von den Steuerzeichen sollten sinnvollerweise nur CR/LF eventuell auch FF vorkommen.

SCRAP.GEM: GEM-Metadateien sollten von allen objektorientierten Zeichenprogrammen unterstützt werden. Aber auch Texteditoren sollten diese einlesen und wenigstens darstellen können (Beispiel: Wordplus unter GEM/3 und X/GEM). Routinen zu ihrer Verarbeitung werden in Kapitel 2 ausführlich dargestellt.

SCRAP.IMG: GEM-Bit-Image-Dateien sollten von allen pixelorientierten Zeichenprogrammen zur Verfügung gestellt werden. Aber auch Texteditoren sollten diese einlesen und wenigstens darstellen können. (Beispiel: Wordplus). Routinen zu ihrer Verarbeitung werden in Kapitel 2 ausführlich dargestellt.

SCRAP.DCA: Die „Document Contents Architecture" ist ein Standardformat von IBM. Leider befindet sich in der Originaldokumentation keine genaue Beschreibung dieses Formats.

SCRAP.USR: OEM-definiert.

Zusätzlich sollte noch mindestens ein weiteres Format unterstützt werden:

SCRAP.DIF: Das DIF-Format dient zum Austausch von Daten zwischen Programmen für Tabellenkalkulation, Präsentationsgrafik und Datenbanken. Ihr Aufbau wird in verschiedenen Literaturstellen aufgezeigt.

Jeder Entwickler hat natürlich auch die Möglichkeit, seine eigenen Formate im Clipboard abzulegen. Wordplus beispielsweise legt zu jedem auf dem Clipboard abgespeicherten Teil eines Dokuments zwei Dateien ab. SCRAP.TXT enthält den ausgeschnittenen Text im ASCII-Format. SCRAP.1WP im 1st Wordplus-Format.

Die Realisierung des Datenaustauschs geschieht nun über die SCRAP-Bibliothek. Dabei stehen folgende Funktionen zur Verfügung:

SCRP_READ: Liest das Inhaltsverzeichnis des Clipboards und gibt es zurück. Zusätzlich werden ab GEM/3 als Funktionswert die Typen der Dateien zurückgeliefert, die sich im Clipboard befinden. Dabei bedeutet ein Rückgabewert von -1, daß kein Inhaltsverzeichnis existiert, ein Wert von 0 zeigt an, daß das Clipboard leer ist, ansonsten gilt:

Bit 0: SCRAP.CSV
Bit 1: SCRAP.TXT
Bit 2: SCRAP.GEM
Bit 3: SCRAP.IMG
Bit 4: SCRAP.DCA
Bit 15: SCRAP.USR

Die entsprechenden Definitionen wurden von uns in die Headerdatei AES.H aufgenommen.

SCRP_WRITE: Setzt das aktuelle Inhaltsverzeichnis für das Clipboard. Dabei wird kein Ordner erzeugt. Dies muß mit Standardbefehlen des Betriebssystems geschehen. Dasselbe gilt für die Dateien im Clipboard, welche mit Standardbefehlen gelesen und geschrieben werden müssen.

Das Inhaltsverzeichnis wird bis GEM 2.X immer so gespeichert, wie es mittels SCRP_WRITE geschrieben wird, d.h. ein

```
scrp_write (''C:\\CLIPBRD'')
```

liefert beim Lesen über „scrp_read" genau

```
C:\CLIPBRD
```

Ein

```
scrp_write (''C:\\CLIPBRD\\'')
```

liefert beim Lesen entsprechend

```
C:\CLIPBRD\
```

Erst ab GEM/3 wird an der letzten Stelle auf jeden Fall ein '\' zurückgeliefert, unabhängig davon, ob dieses Zeichen beim „scrp_write" angegeben wurde. Wir können deshalb nur empfehlen, auf jeden Fall den letzten '\' anzugeben, um kompatibel zu bleiben.

SCRP_CLEAR: Ab GEM 2.X gibt es auch einen Befehl zum Löschen der Dateien im Clipboard. Für den ST wird dieser Befehl im Modul CLIPBRD (Kapitel 6) nachgebildet.

Bei GEM/3 wird auch beim ersten SCRP_READ ein korrekter Pfad zurückgeliefert, z.B. C:\GEMAPPS\CLIPBRD\, falls sich das System auf Laufwerk C befindet. Das bedeutet, daß alle Programme durch einen entsprechenden SCRP_READ-Befehl dieses Inhaltsverzeichnis geliefert bekommen und dadurch der Datenaustausch vollständig gewährleistet ist.

Leider gilt dies nicht für ältere GEM-Versionen. Obwohl bereits ein Ordner vorgesehen war (GEMSCRAP) bringt ein SCRP_READ-Befehl kein Ergebnis. Dies bedeutet, daß vorher ein SCRP_WRITE geschehen muß. Nun kommt das Problem: Wenn jedes Programm ein SCRP_READ benutzt, um den Ordner des Clipboards festzustellen und bei nichtvorhandenem Ordner ein SCRP_WRITE, um ihn zu setzen, dann kann es sein, daß sich sehr viele Ordner auf der Diskette/Festplatte befinden. Dabei wird aber immer jeweils nur einer als aktuelles Clipboard benutzt. Aus diesem Dilemma gibt es nur einen Ausweg: Jeder Entwickler muß den gleichen Namen für das Clipboard benutzen. Wir empfehlen den bereits für GEM/3 und durch Wordplus auch auf dem Atari eingeführten Namen: CLIPBRD.

Ein weiteres Problem ergibt sich für das Laufwerk, auf dem sich der Ordner befinden muß. Bei GEM/3 ist dies kein Problem, da hier der SCRP_READ-Befehl auch das Laufwerk beinhaltet. Beim Atari oder bei GEM 2.X muß jedoch irgendein Laufwerk bestimmt werden. Am besten ist dies ein „schnelles" Laufwerk, also eine Festplatte oder eine RAM-Disk. Beim IBM PC kann das Laufwerk C, also die Festplatte benutzt werden, da GEM sowieso nur auf Festplatte einen Sinn macht.

Folgender Algorithmus wird nun angewandt, um das Inhaltsverzeichnis des Clipboards zu bestimmen:

1. Ein „scrp_read" bestimmt, ob bereits von irgendeinem Programm ein Inhaltsverzeichnis gesetzt wurde. Dieses wird dann benutzt. Ab GEM/3 und X/GEM wird immer ein gültiges Inhaltsverzeichnis gefunden.

2. Ist keines gesetzt, so wird beim ST nachgeprüft, ob es ein schnelles Laufwerk gibt (ab C aufwärts). An dieses wird der Ordner CLIPBRD angehängt. Gibt es kein schnelles Laufwerk, so wird Laufwerk A benutzt. Beim PC wird das Laufwerk C benutzt. Anschließend wird geprüft, ob sich bereits ein Ordner dort befindet. Ist dies der Fall, wird der Ordner als Clipboard weiterbenutzt.

3. Existiert kein Ordner, so wird ein Ordner erzeugt. Ist das Erzeugen ordnungsgemäß vonstatten gegangen, wird der Ordner benutzt.

4. Kann der Ordner nicht erzeugt werden, so kann auch das Clipboard nicht benutzt werden, d.h. die Menüpunkte, die es betreffen, müssen abgeschaltet werden.

Eine elegante Lösung, die Schritte 2, 3 und 4 zu übergehen, ist die Möglichkeit von Accessories, ebenfalls einen Clipboard-Ordner zu erzeugen. Wenn Accessories nach oben

erwähntem Algorithmus vorgingen, um einen Ordner zu erzeugen und einen „scrp_write"-Befehl ausführen würden, so würden Applikationen immer schon einen gültigen Pfad finden.

In dem in Kapitel 6 beschriebenen Beispielprogramm wird im Modul GLOBAL die oben beschriebene Methode benutzt. Außerdem wird der fehlende SCRP_CLEAR-Befehl im Modul CLIPBRD implementiert sowie der Rückgabewert des SCRP_READ-Befehls an GEM/3 angepaßt.

Wie sollten nun Programme vorgehen, um den Datenaustausch möglich zu machen? Zuerst wird wie oben beschrieben das Inhaltsverzeichnis angelegt.

Beim Kopieren (Copy) in das Clipboard wird nun entsprechend den angegebenen Standardformaten jeweils eine Datei SCRAP.??? erzeugt. Die drei Fragezeichen stehen für die jeweilige Endung. Bei Texteditoren bietet sich für ASCII-Text SCRAP.TXT an. SCRAP-Dateien mit eigener Endung wie SCRAP.1WP oder SCRAP.SDO können selbstverständlich ebenso zusätzlich geschrieben werden. Für objektorientierte Zeichenprogramme gilt das gleiche für Metadateien mit Endung GEM, für pixelorientierte Programme entsprechendes für Bit-Image-Dateien mit Endung IMG.

Beim Ausschneiden (Cut) wird selbstverständlich die gleiche Methode angewandt. Lediglich das Original wird dabei zusätzlich gelöscht.

Beim Einfügen (Paste) aus dem Clipboard wird zuerst gesucht, ob sich eine entsprechende Datei überhaupt dort befindet. Nur dann ist der Menüpunkt anwählbar. Dann wird die Datei aus dem Inhaltsverzeichnis gelesen. Dies kann auch mehrere Male geschehen.

Ein weiteres wichtiges Kriterium zum Thema Datenaustausch betrifft den Aufruf anderer Applikationen aus einer bestehenden Applikation. Beispielsweise möchte man ein Textprogramm verlassen, weil man die Grafik, die man eben eingebunden hat, noch einmal überarbeiten möchte. Am besten wäre ein Aufruf des Grafikprogrammes, an welches man als Parameter die Grafik übergibt, die man bearbeiten möchte. Nach Bearbeiten der Grafik sollte man sich wieder im Textprogramm befinden.

Beim ST geht dies am einfachsten, wenn man andere Programme über die GEMDOS-Funktion „Pexec" aufruft. Dann ist man nach der Rückkehr automatisch an der richtigen Stelle im aufrufenden Programm. Reicht der Speicher aber nicht aus, was bei IBM PCs praktisch immer der Fall ist, so muß man einen „shel_write" Befehl benutzen. Das aufrufende Programm ist dann aber nicht mehr im Speicher.

Die einzige Rettung besteht darin, dem aufgerufenen Programm mitzuteilen, von wo es aufgerufen wurde. Bei den GEM-Versionen auf dem IBM PC wird dies bei allen Programmen gemacht, welche das OUTPUT.APP aufrufen. Von diesem Programm kann man dann wieder das aufrufende Programm starten. Mitgeteilt wird dies dem OUTPUT.APP durch eine standardisierte Nachricht beim „shel_write"-Befehl. Dabei wird

neben dem Parameter auch der Name des aufrufenden Programms, getrennt durch einen '/', übergeben. Für GEMDRAW, welches DRAW.APP heißt und der Grafik TEST.GEM heißt, der Parameter „tail" im „shel_write":

TEST.GEM/DRAW.APP

Das Programm OUTPUT.APP extrahiert hiervon den Parameter TEST.GEM und das aufrufende Programm DRAW.APP. Dieses kann dann wieder von OUTPUT.APP aufgerufen werden, so daß sich die Wirkung eines Unterprogrammes ergibt. Dadurch ist ein Wechsel von einem Programm in ein anderes möglich, welches wiederum das aufrufende Programm ausführen kann. OUTPUT.APP ruft dann ein „shel_write" mit

TEST.GEM/OUTPUT.APP

auf. Für GEM 2.X und neuere Versionen gibt es allerdings noch eine andere Möglichkeit: Die Befehle „shel_rdef" und „shel_wdef" verwalten das Standardprogramm, welches nach Verlassen eines Programmes aufgerufen wird. Damit kann verhindert werden, daß nach Beendigung eines Programms der Desktop zum Zuge kommt. Stattdesssen wird das durch „shel_wdef" definierte Programm gestartet.

Als Regel sei an dieser Stelle erwähnt, sicherheitshalber einen vorhandenen '/' im Parameterteil vom eigentlichen Parameter abzutrennen, um den Namen des aufrufenden Programms herauszubekommen. Existiert kein '/', so handelt es sich um den Desktop oder ein anderes Programm, welches seinen Namen nicht an den Parameter anhängt. Auf jeden Fall gilt diese Regel für alle Programme, die nicht nur vom Desktop aus aufgerufen werden können, wie z.B. OUTPUT.APP.

Sollen mehrere Parameter an ein Programm übergeben werden, so müssen diese, wenn sie über „shel_read" gelesen werden, durch ein Trennzeichen getrennt werden. Der Desktop von GEM 2.X und späteren Versionen übergibt beispielsweise an OUTPUT mehrere GEM-Dateien, wenn man diese zuerst anklickt und dann den Menüpunkt „An Ausgabe" anwählt. Die Dateinamen werden dabei durch Kommata getrennt. Wird nur an den ersten der komplette Pfad angehängt, so befinden sich die anderen Dateien auf dem gleichen Inhaltsverzeichnis. Möchte man noch den Namen des aufrufenden Programms einbringen, so muß dies hinter dem letzten Parameter geschehen. OUTPUT erlaubt aber auch die Übergabe von mehreren Parametern, welche durch ein Leerzeichen getrennt sind. So sind gültige Beispiele für Parameter:

C:\GEMAPPS\TEST1.GEM,TEST2.GEM

C:\GEMAPPS\TEST1.GEM TEST2.GEM

C:\GEMAPPS\TEST1.GEM,TEST2.GEM/DRAW.APP

## 5.6 Ausgabe von Daten

Ein immer wiederkehrendes Problem ist die Ausgabe von Daten auf dem Drucker oder einem anderen Ausgabegerät. Jedes Programm hat seine eigenen Möglichkeiten vorgesehen. So muß bei Wordplus eine entsprechende Konfigurationsdatei geladen werden, bei Signum gibt es verschiedene Programme für verschiedenartige Drucker usw.

Dem Benutzer ist bei dieser Situation nicht unbedingt geholfen. Er muß für jedes Programm immer wieder Druckerinstallationen vornehmen. Meist hat er einen oder zwei verschiedene Drucker an seinem System angeschlossen. Hervorragend wäre nun die Möglichkeit, diese ein für allemal festzulegen (zumindest bis zu einem neuen Druckerkauf), und die einzelnen Programme würden über eine Standardschnittstelle an die entsprechenden Geräte ausgeben.

Bei GEM auf dem PC ist dieser Weg gegangen worden. Dort wird das GEM für eine bestimmte Konfiguration eingestellt. Einmal konfiguriert, kann nun der Benutzer bestimmen, auf welches Ausgabegerät seine Daten geschickt werden sollen. Dabei werden Standardformate verwendet, die jedes Programm erzeugen kann. Ein entsprechendes Ausgabeprogramm, nämlich OUTPUT.APP gibt dann die entsprechenden Dateien auf das Ausgabegerät aus.

Der Vorteil von OUTPUT liegt in dessen freier Konfigurierbarkeit. Beim Installieren von GEM können nahezu beliebige Ausgabegeräte eingestellt werden wie Matrixdrucker, Laserdrucker, Plotter oder Kamera. Wird an OUTPUT eine Metadatei übergeben, so wird sie anstandslos auf dem eingestellten Ausgabegerät ausgegeben. Das gleiche gilt für Bit-Image-Dateien.

Für ASCII-Texte wird ebenfalls ein Standardformat benutzt. Es handelt sich um Dateien mit dem Suffix OUT. Dieses Format wurde in Kapitel 2 beschrieben. Tatsächlich arbeitet die neue Version von Wordplus auf dem PC mit OUTPUT zusammen. Dabei wird statt dem direkten Druck eine entsprechende OUT-Datei erzeugt und diese an OUTPUT übergeben. Dort kann man dann bestimmen, auf welches Ausgabegerät gedruckt werden soll. Sogar ein Drucken im Hintergrund ist mit Hilfe des Spoolers CALCLOCK möglich.

Die Definitionen der Einstellungen in der OUT-Datei sind in der Headerdatei VDI.H aufgeführt. Dort sind alle Einstellungen zum Drucken von Texten mit sämtlichen Attributen, also fett, kursiv, unterstrichen, schmal, breit usw. aufgeführt.

Auf dem ST mit seinem großen Speicher könnte man ohne Probleme ebenfalls ein solches Programm schreiben, das auch als Accessory im Hintergrund drucken könnte. Die Anpassung an das jeweilige Ausgabegerät wird dann von diesem einen Accessory ein für allemal für alle Applikationen übernommen. Das würde bedeuten, daß der Benutzer nur noch einmal seine Konfiguration angibt. Die Applikationen erzeugen dann entweder OUT-, GEM- oder IMG-Dateien. In den OUT-Dateien gibt es auch Befehle, um Metadateien und Bit-Image-Dateien zu drucken, so daß Text und Grafik gemischt werden können.

Für alle länger andauernden Routinen (vor allem Ausgaberoutinen) sollte außerdem gelten, daß sie jederzeit durch eine Taste, am besten „Esc" und/oder „Undo" (nicht auf dem PC) oder auch z.B. durch Drücken beider Shift-Tasten, abgebrochen werden können.

Dieses Konzept des universellen Ausgabeprogramms ist zumindest für alle Programme interessant, die nur Text oder Grafik in den Standardformaten ausgeben, also z.B. für alle Textverarbeitungs-, Datenbank- und Tabellenkalkulationsprogramme. Die Ausgaberoutinen für Meta- und Bit-Image-Dateien wurden bereits in Kapitel 2 beschrieben, ein Interpretierer für das OUT-Format ist ebenso leicht zu schreiben.

## 5.7 Allgemeines

An dieser Stelle sollen noch einige allgemeine Tips zur Standardisierung gegeben werden.

– Bildschirmauflösung

Es gibt immer noch viele Programme, welche die Auflösung des Bildschirms mittels Getrez() erfragen. Stellt sich heraus, daß es sich um eine niedrige Auflösung handelt, so läuft das Programm deswegen nicht los, weil es davon ausgeht, daß die Menüzeile nicht auf den Bildschirm paßt. Nicht nur seit der Zeit als Großbildschirme eingeführt wurden, sondern spätestens bei der Einführung des Atari TT machen diese Programme eine schlechte Figur. Viele Farben bedeuten noch lange nicht, daß nur 40 Zeichen in einer Zeile dargestellt werden können.

Entwickler können bereits heute ihre Programme auf große Auflösung testen, indem sie softwaremäßig emulierte Großbildschirme einsetzen, wie z.B. „Bigscreen" von Julian Reschke. Viele Programme laufen nicht im 640x400 Modus mit 16 Farben, obwohl sie es könnten. Dies gilt natürlich nicht für in Assembler geschriebene Spiele, die ihre Ausgabe direkt ins Video-Ram schmieren. Aber solche Programme laufen mit Sicherheit auch nicht auf dem TT.

– Vektorverbiegende Programme

Für vektorverbiegende Programme wurde inzwischen das XBRA-Protokoll (eXtended BRAner von Moshe Braner) erfolgreich von Julian Reschke eingeführt (siehe auch verschiedene Artikel im ST Magazin). Damit können Programme leichter feststellen, ob sie schon installiert sind und können sich leichter aus der Vektorkette ausklinken. Auch wenn dies wohl ein Verfahren ist, das nur bei den wenigsten Programmen zum Einsatz kommt, sollten alle Programme, die es betrifft, zum Schutz des Benutzers dieses Standardverfahren einsetzen.

– Speicherplatz

Jede Applikation sollte nur soviel Speicherplatz allozieren, wie sie wirklich benötigt. Dies gilt vor allem für Accessories und Programme im Auto-Ordner.

– Ausgabegröße

Jede Applikation sollte, soweit die Ausgabegröße bekannt oder berechenbar ist, vor dem Sichern den freien Speicherplatz auf dem gewählten Speichermedium abfragen und ggf. eine Umlenkung vorschlagen.

Da das Abfragen des freien Speicherplatzes von Festplatten bei älteren TOS-Versionen etwas länger dauert, Festplatten aber wiederum selten voll beschrieben werden, kann sich die Abfrage im Prinzip auf die Laufwerke A und B beschränken.

# 6 Ein universelles Modulkonzept in C

In diesem Kapitel soll erklärt werden, wie man in der Sprache C Module schreibt, die möglichst unabhängig voneinander sind. Sprachen wie z.B. Modula-2 unterstützen dieses Konzept sehr stark. Modula ist deshalb auch die Sprache, welche man als echte Alternative zu C lernen sollte. Doch auch in C ist es möglich, Module mit genau definierten Schnittstellen zu entwickeln.

Seit Einführung der ANSI-C-Compiler wird diese Entwicklung sogar noch begünstigt, da nun auch Parameter von Funktionen auf ihre Gültigkeit überprüft und gegebenenfalls angepaßt werden. Die Funktionalität eines ANSI-C-Compilers reicht sogar soweit, daß man gewarnt wird, wenn man eine Funktion aufrufen möchte, die nicht vorher definiert wird. Dadurch ergibt sich ein ähnlicher Effekt wie bei Modula oder auch Pascal. Bei der Übersetzung von Programmen, in welchen Funktionen, die nicht definiert sind, aufgerufen werden, wird sofort eine Fehlermeldung ausgegeben.

Wir werden auf den Aufbau von Modulen in C in 6.2 zu sprechen kommen und erklären, warum es so wichtig ist, Programme in Module zu unterteilen.

## 6.1 Ziel der Beispielapplikation SCRAP

Bevor wir uns auf die einzelnen Module der Beispielapplikation stürzen, sollen zunächst die Ziele vorgestellt werden, die wir verfolgen. Von diesen Zielen ausgehend, kann man dann die einzelnen Aufgaben definieren und auf Module verteilen.

Die Applikation wird SCRAP heißen und soll die Verwaltung eines System-Klemmbretts bereitstellen. Das System-Klemmbrett ist ein Inhaltsverzeichnis auf der Festplatte oder Diskette. In dieses können Programme kleine Stückchen, auch Scraps genannt, hineinschreiben oder auslesen. Die Dateien, die sich in diesem Inhaltsverzeichnis befinden, heißen alle SCRAP, haben jedoch einen spezifischen Suffix (z.B. SCRAP.GEM). An diesem erkennt man, um welche Art von Datei es sich handelt.

Die Applikation soll wiederum ein solches Inhaltsverzeichnis suchen. Wenn noch keines existiert, soll es dieses anlegen. Dann wird das Inhaltsverzeichnis in einem Fenster mit Hilfe von Piktogrammen angezeigt. Auf Wunsch sollte auch Textdarstellung möglich sein, ähnlich wie im Desktop. Das verschiedene Aussehen der Piktogramme spiegelt die Art wider, um die es sich handelt. In Abbildung 6.1 wird ein Beispiel für ein Fenster mit verschiedenen Piktogrammen gezeigt.

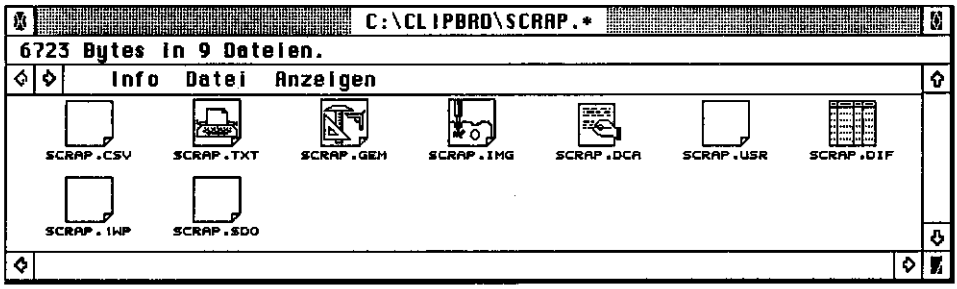



**Abbildung 6.1:** Piktogramme der Scrap-Directory

Die Piktogramme sollen nun angeklickt und geöffnet werden können. Dann erscheint je nach Art des Piktogramms ein Fenster, welches den Inhalt der Datei anzeigt. Das bedeutet, daß je nach Art ein anderer Algorithmus angewandt werden muß. Drei Arten werden vom Programm verarbeitet. Metadateien, Bit-Image-Dateien und normaler ASCII-Text. Die Datei mit Suffix GEM wird als Metadatei, die mit Suffix IMG als Bit-Image-Datei und alle anderen werden als Textdatei geöffnet.

Die Bedienung des Klemmbretts selbst soll in einer Umgebung geschehen, welche einen eigenen Desktop zur Verfügung stellt. Dies ist nicht unbedingt nötig, aber das Programm soll einige Techniken aufzeigen, wie man einen eigenen Desktop verwaltet.

Darüberhinaus sollen zwei weitere Arten von Fenstern geöffnet werden können, die aber nur der Demonstration dienen sollen. Es handelt sich zum einen um eine Art von Fenster, welche Potenzen einer Zahl zwischen zwei und neun darstellt. Dort werden die Pop-Up-Menüs und ein eigener Mauscursor veranschaulicht. Zum anderen sollen Fenster geöffnet werden, welche eine Ellipse mit verschiedenen Mustern zeichnet, die der Benutzer interaktiv einstellen kann. An diesem Fenster wird eine Art Multitasking-Betrieb veranschaulicht.

Die Bedienung soll intuitiv und einheitlich sein. So kann man jedes Objekt jederzeit anwählen und eine Info- oder Hilfe-Tafel einblenden. Dies soll auch bei nicht aktiven Objekten geschehen können. Die Information oder Hilfe bezieht sich dann auf das oberste Fenster.

Am Clipboard-Fenster wird außerdem eine eigene Menüzeile demonstriert.

Das ganze Programm soll darüberhinaus überall dort lauffähig sein, wo GEM zur Verfügung steht. An den Quelltexten soll kein einziges Byte geändert werden müssen, wenn es auf einen anderen Rechner oder ein anderes GEM übertragen wird. Es soll lediglich neu compiliert und gelinkt werden. Das Programm soll soweit auch betriebssystem- und rechnerunabhängig sein. Außerdem soll es auch als Accessory im Hintergrund laufen können, ohne daß dabei die Funktionalität darunter leidet. Ähnliches soll für die Auflösung gelten. Es soll unerheblich sein, ob es auf dem Atari unter den drei verschiedenen Auflösungen läuft oder auf einem PC mit einer entsprechenden Grafikkarte. Ein weiterer Punkt ist die Compilerunabhängigkeit. Sie sollen sich für Ihren Lieblingscompiler entscheiden können. Er sollte allerdings nicht allzu fehlerhaft sein, wie dies Vorversionen bestimmter Compiler schon waren. Wir können aber nur eine Garantie auf Lauffähigkeit für die in Kapitel drei aufgeführten Compiler übernehmen, da diese getestet wurden.

Falls das Programm als Accessory gestartet wird (im Prinzip nur sinnvoll auf einem Atari ST), so soll es, um keine Funktionalität einzubüßen, den Desktop in ein Fenster legen, da dieser der Hauptapplikation gehört. Ähnliches gilt für die Menüzeile. Sie soll sich dann innerhalb des Desktop-Fensters befinden und nach Anklicken genauso zu bedienen sein wie die Hauptmenüzeile.

## 6.2 Aufbau und Aufgaben der Module

Um zu verstehen, worum es in diesem Abschnitt geht, sollte zunächst einmal der Begriff des Moduls definiert werden:

Unter einem Modul verstehen wir einen klar umrissenen, in sich abgeschlossenen Teil eines Programmsystems, der eine funktional spezifizierte Teilaufgabe implementiert. Die Beschreibung der Modulfunktion bildet seine Schnittstelle. Über diese tritt ein Modul mit seiner Umgebung in Beziehung.

Eine Schnittstelle muß möglichst unabhängig von den Algorithmen sein, mit denen ein Modul implementiert ist. Dieses Prinzip nennt man auch „Information Hiding", da die internen, die Implementierung betreffenden Einzelheiten lokal gehalten und nicht an die Umgebung weitergegeben werden.

Bei der Definition der Schnittstelle sollte darauf geachtet werden, daß diese so minimal wie möglich gehalten wird. Damit vermindert man den Kommunikationsaufwand, vermeidet Mißverständnisse und verhindert letztlich Fehler, die aufgrund falscher Annahmen bei umfangreichen Schnittstellen sehr leicht entstehen können.

Beispiele für Module sind unter anderem die mathematischen Funktionen wie „sin(x)". Für jeden Wert „x" wird hier eine Ausgabe geliefert. Die Schnittstelle besteht nur aus der einen Funktion und der Vorgabe, wie das x und der Funktionswert zu interpretieren sind. Wie die Funktion „sin(x)" nun den Sinus genau berechnet, also über Tabellen oder durch Berechnung der Potenzreihe, interessiert den Aufrufer nicht. Wichtig ist, was hin-

eingesteckt wird und was danach herauskommt. Außerdem muß die Sinus-Funktion nicht neu übersetzt werden, wenn das aufrufende Modul verändert wird. Es wurde einmal übersetzt und liegt damit jederzeit bereit, zu einem Programm hinzugebunden zu werden.

In der Sprache Modula-2 werden Module durch eine Schnittstelle und eine Implementierung beschrieben. Dort gibt es zwei getrennte Dateien, nämlich mit Endung DEF für die Definition und mit Endung MOD für die Implementierung. Möchte nun ein Modul Funktionen aufrufen, welche in einem anderen Modul definiert sind, so muß es diese importieren. Umgekehrt kann ein Modul Funktionen aber auch Konstanten, Typen und Variablen exportieren und sie so anderen Modulen zugänglich machen. Was ein Modul nicht exportiert, kann nicht von einem anderen Modul benutzt werden.

Eine wichtige Eigenschaft von Modulen ist auch die der getrennten Übersetzbarkeit. Dies hängt natürlich von der gewählten Sprache ab. In Modula, aber auch in C ist dies ohne weiteres möglich.

Ein Beispiel bei Verwendung der Funktion „WriteInt" aus dem Modul „InOut":

```
MODULE Test;

FROM InOut IMPORT WriteInt; (* Damit ist die Funktion sichtbar *)

BEGIN
  WriteInt (3, 10);
END Test.
```

Mit diesem Programm wird die Zahl drei auf zehn Stellen rechtsbündig ausgegeben. Würde die Zeile „FROM..." fehlen, so würde der Compiler bei Erreichen der Zeile „WriteInt(3, 10)" feststellen, daß er die Funktion nicht kennt und sich weigern, Code zu erzeugen und damit das Programm ablauffähig zu machen. Und das ist auch gut so. Durch den Import der Funktion kennt der Compiler die Parameter und kann nun leicht testen, ob zum Beispiel der zweite Parameter, der die Feldbreite angibt, fehlt oder ob die Typen der Parameter übereinstimmen.

In der Sprache C war es bisher üblich, Funktionen einfach aufzurufen und dem Linker die Sorge zu überlassen, wo er die Funktion finden und dazubinden soll. Was die Parameter angeht, so konnte man bisher ohne Probleme auch mal mehr oder weniger Parameter an eine Funktion übergeben. Das bedeutete aber, daß zur Laufzeit das Programm mit großer Wahrscheinlichkeit abstürzte oder zumindest merkwürdige Ergebnisse zur Folge hatte.

Unser Beispiel von oben würde in Standard-C etwa so aussehen:

```
main ()
{
  printf ("%10d", 3);
} /* main */
```

Diese Zeilen würden ebenfalls die Zahl 3 auf 10 Stellen rechtsbündig ausgeben. Weder der Compiler noch der Linker hätten allerdings etwas dagegen, wenn wir den „printf"-Befehl in

```
printf (3, 10);
```

umwandeln würden.

Anders sieht dies bei den modernen ANSI-C-Compilern aus. Sie wollen für jede Funktion auch einen Prototyp. Der Prototyp bezeichnet die Funktion mit ihren Parametern. Standardfunktionen wie „printf", die sich in jeder C-Bibliothek befinden, haben genauso Prototypen wie eigene Funktionen. Wird nun obiges Programm beispielsweise mit dem Turbo-C-Compiler übersetzt, dann würde er eine Warnmeldung bringen, daß der Prototyp von „printf" fehlt. Diesen benötigt er einerseits für die Parameterüberprüfung, andererseits dafür, sich auszurechnen, wieviele Parameter er in Registern übergeben kann. Um den Prototyp sichtbar zu machen, muß eine Datei, in der ein solcher Prototyp steht, in das Programm aufgenommen werden. Dies geschieht bekanntlich mit der „include"-Direktive. Wir würden also unser Programm erweitern, indem wir als erstes ein

```
#include <stdio.h>
```

einfügen.

Daraufhin wird der C-Compiler zufrieden sein, da sich der Prototyp für die Funktion „printf" in der Datei „stdio.h" befindet. Er lautet

```
int printf (const char *format, ...);
```

Das bedeutet, daß ein „printf"-Aufruf immer zuerst eine Zeichenkette erwartet, dann eine beliebige Anzahl von Parametern (sie hängen von der Zeichenkette ab).

Ist ein solcher Prototyp einmal definiert, wird der Compiler auch unseren falschen

```
printf (3, 10);
```

Befehl nicht annehmen. Wir sind also vor unliebsamen Überraschungen sicher. Die Funktion „printf" befindet sich im Modul „stdio" in einer Bibliothek. Eine solche Bibliothek verfügt meist über sehr viele Funktionen, die dann in den entsprechenden Header-Dateien (Suffix „H"). beschrieben werden. Diese Module werden von den Compiler-Entwicklern mitgeliefert. Um zu erreichen, daß Programme mit vielen Compilern lauffähig sind, muß man bei der Wahl der Funktionen darauf achten, daß diese in jeder Compiler-Bibliothek vorhanden sind. Glücklicherweise gibt es eine Reihe von Funktionen, die bei jedem Compiler die gleichen Effekte haben.

Um nun eine Schnittstelle in C zu erstellen, werden wir ebenfalls für jedes Modul zwei Dateien erstellen. Die Schnittstellenbeschreibung wird in der Datei mit Suffix „H" eingetragen, der eigentliche Programmcode in der Datei mit Suffix „C", also z.B. GLOBAL.H und GLOBAL.C.

In der Header-Datei werden alle Konstanten, Typen, Variablen und Funktionen eingetragen, die von den anderen Modulen importiert werden können, also von anderen Modulen aus sichtbar sind. Beispiele sind die Dateien auf der mitgelieferten Diskette, die mit Suffix „H" versehen sind.

Ein Problem ergibt sich nun, wenn Variablen, die in einer Header-Datei definiert sind, sowohl von anderen (importierenden) Modulen, als auch vom exportierenden Modul benötigt werden. Variablen, die in C zwar angesprochen werden können, jedoch keinen Speicherplatz belegen, sind externe Variablen. Sie werden in irgendeinem anderen Modul definiert, wo sie auch Speicherplatz zugewiesen bekommen. Im folgenden Beispiel sei „gl_apid" zunächst eine Variable, die von mehreren Modulen benutzt wird, aber nur in einem Modul Speicherplatz belegt. In diesem (exportierenden) Modul (Modul1) wird die Definition und ein Benutzen der Variablen z.B. folgende Gestalt annehmen:

```
/***** MODUL1.C *****/

#include <portab.h>

WORD gl_apid;
WORD wert;

LOCAL VOID test1 ()

{
  gl_apid = appl_init ();
  wert    = 5;
  ...
} /* test1 */
```

In einem anderen Modul (Modul2) könnte die Benutzung so aussehen:

```
/***** MODUL2.C *****/

#include <portab.h>

EXTERN WORD gl_apid;

VOID test2 (x)
WORD x;

{
  WORD apid;

  apid = gl_apid;
  ...
} /* test2 */
```

Wir sehen, daß die Variable einmal normal definiert ist (sie bekommt dann ihren Speicherplatz zugewiesen) und einmal als EXTERN, ein Bezeichner, der in der Datei PORTAB.H als „extern" definiert ist, d.h durch den Befehl

```
#define EXTERN extern
```

Nun tritt folgendes Problem auf: Wenn die Variable ihren Namen (gl_apid) oder ihren Typ (WORD) ändert, so muß diese Namensänderung in allen Dateien, die diese Variable benutzen, ebenfalls durchgeführt werden, und zwar in der Extern-Deklaration und im Programmcode. Dies gilt natürlich für alle Variablen, das können durchaus einige Dutzend sein.

Wir trennen nun zunächst die Definition der Variablen, Konstanten und Typen von der eigentlichen Implementierung. Dadurch ergibt sich für obiges Beispiel folgendes Bild für Modul1:

```
/***** MODUL1.H *****/

WORD gl_apid;

/***** MODUL1.C *****/

#include <portab.h>

#include "modul1.h"

WORD wert;

LOCAL VOID test1 ()

{
  gl_apid = appl_init ();
  wert    = 5;
  ...
} /* test1 */
```

Für Modul2 benötigen wir momentan nur die C-Datei:

```
/***** MODUL2.C *****/

#include <portab.h>

#include "modul1.h"

VOID test2 (x)
WORD x;

{
  WORD apid;

  apid = gl_apid;
  ...
} /* test2 */
```

Der Unterschied zur ersten Ausführung besteht nun darin, daß die Definition von „gl_apid" nur noch einmal auftritt, nämlich in der Headerdatei MODUl1.H. Modul1 hat einen „include"-Befehl, um die Definition von „gl_apid" zu kennen. Das gleiche gilt für MODUL2. Ganz richtig ist die zweite Ausführung jedoch nicht. Denn durch die „include"-Anweisung in Modul2 wird ebenfalls Speicherplatz für „gl_apid" belegt, da diese Definition nicht das Schlüsselwort „extern" benutzt.

Ein Ausweg aus diesem Dilemma finden wir, indem wir Variablen noch ein Schlüsselwort voranstellen, welches von beiden Modulen verschieden interpretiert wird. Dieses Schlüsselwort heißt GLOBAL und wird einmal als

```
#define GLOBAL EXTERN
```

und ein anderes Mal als

```
#define GLOBAL
```

definiert. Diese Definitionen werden in einer speziellen Headerdatei untergebracht. Eine Datei heißt „import.h" und benutzt die erste Definition (EXTERN), die andere Datei heißt „export.h" und benutzt die zweite Definition. Unsere Module von oben haben nun folgendes Aussehen:

```
/***** MODUL1.H *****/

GLOBAL WORD gl_apid;

/***** MODUL1.C *****/

#include <portab.h>

#include "export.h"
#include "modul1.h"

LOCAL WORD wert;
LOCAL VOID test1 ()

{
  gl_apid = appl_init ();
  wert    = 5;
  ...
} /* test1 */
```

Für Modul2 lautet die C-Datei:

```
/***** MODUL2.C *****/

#include <portab.h>
```

```
#include "import.h"
#include "modul1.h"

VOID test2 (x)
WORD x;

{
  WORD apid;

  apid = gl_apid;
  ...
} /* test2 */
```

Wie wir nun leicht einsehen, werden von Modul1 alle Variablen, die sich in der Datei MODUL1.H befinden, exportiert. Durch Benutzung des Befehls

```
#include ''export.h''
```

wird die Definition von GLOBAL als leer angenommen, woraus sich innerhalb von MODUL1.H ein

```
WORD gl_apid;
```

ergibt. Anders sieht es bei Modul2 aus. Dort wird durch Benutzung des Befehls

```
#include ''import.h''
```

die Definition von GLOBAL als EXTERN (= extern) angenommen, woraus sich innerhalb von MODUL1.H ein

```
EXTERN WORD gl_apid;
```

ergibt. Dies hat nun schon eine gewisse Ähnlichkeit mit Modula-2, wo man ja auch alle Funktionen, Variablen und Typen, die man in einem Modul benutzen möchte, zuerst importiert. In C entspricht der „IMPORT"-Befehl von Modula nun der Einbindung von „import.h".

Um Namensgleichheiten beim Linken über mehrere Module so gering wie möglich zu halten, sollten möglichst wenige Bezeichner jeweils exportiert werden. Alle Bezeichner, die nur innerhalb eines Moduls benötigt werden, sollten auch nur dort definiert sein. Man erreicht dies durch Voranstellen des Standard-C-Bezeichners „static". Dieser Bezeichner wird auch in der Datei PORTAB.H mittels

```
#define LOCAL static
```

und

```
#define MLOCAL static
```

definiert. Das „M“ soll für Modul stehen. Da man mit „lokal“ aber immer lokal zu einem Modul meint, kann man das „M“ auch weglassen. Man hat dann die beiden Bezeichner „GLOBAL“ für globale Vereinbarungen, die überall sichtbar sind, und „LOCAL“ für lokale Vereinbarungen, die nur innerhalb eines Moduls ihre Gültigkeit haben. Der große Vorteil der lokalen Definitionen besteht darin, daß Bezeichner in verschiedenen Modulen die gleichen Namen haben können, da der Compiler die lokalen Definitionen vorzieht.

Bevor wir unser Beispiel von oben erweitern, wollen wir noch das eingangs Gesagte benutzen. Demnach besitzt jedes Modul sowohl eine Schnittstellenbeschreibung als auch eine Implementierung. Die beiden Module von oben und das „import.h“ sowie „export.h“ haben nun folgende Form:

```
/***** IMPORT.H *****/

#include <portab.h>
#include <aes.h>
#include <vdi.h>

#define GLOBAL EXTERN

/***** EXPORT.H *****/

#undef GLOBAL
#define GLOBAL

/***** MODUL1.H *****/

#define MAX 10

GLOBAL WORD array [MAX];
GLOBAL WORD gl_apid;

/***** MODUL1.C *****/

#include "import.h"

#include "export.h"
#include "modul1.h"

LOCAL WORD wert;

LOCAL VOID test1 _((VOID));

LOCAL VOID test1 ()

{
  gl_apid = appl_init ();
  wert    = 5;
  ...
} /* test1 */
```

```
/***** MODUL2.H *****/

GLOBAL VOID test2 _((WORD x));

/***** MODUL2.C *****/

#include "import.h"
#include "modul1.h"

#include "export.h"
#include "modul2.h"

LOCAL WORD wert;

GLOBAL VOID test2 (x)
WORD x;

{
  WORD apid, i;

  wert = x;
  apid = gl_apid;
  i    = 0;

  while (i < MAX) array [i++] = x;
  ...
} /* test2 */
```

IMPORT.H und EXPORT.H wurden schon oben erklärt. In IMPORT.H wurden noch die Dateien „portab.h", „aes.h" und „vdi.h" eingebunden.

Jedes Modul besteht nun aus den zwei Dateien, die jeweils ein Suffix „H" oder „C" tragen. In MODUL1.H wird die Konstante MAX sowie die globalen Variablen „array" und „gl_apid" definiert. Auf diese Bezeichner kann von jedem Modul aus zugegriffen werden, welches diese Datei durch den entsprechenden „include"-Befehl importiert.

MODUL1.C beginnt zunächst mit den „include"-Anweisungen. Zuerst wird die Datei „import.h" eingebunden. Da aber nichts importiert wird, folgt nun keine weitere Include-Datei. Immerhin können wir auch auf die Einbindung von „portab.h" verzichten, da wir diese ins „import.h" verschoben haben. Danach wird die Datei „export.h" eingebunden und damit die eigene Headerdatei exportiert.

Nach den „include"-Befehlen werden nun die Modul-internen, d.h. lokalen Variablen aufgeführt. Durch das Zusatzwort LOCAL (= static) wird der Bezeichner (hier „wert") nicht nach außen getragen. Damit kann in anderen Modulen derselbe Bezeichner verwendet werden. Es könnte immerhin sein, daß ein fremdes Modul dazugebunden werden muß. Da man möglicherweise nicht alle Namen aus diesem anderen Modul kennt, ist es besser, wenn jedes Modul so wenig Namen wie möglich nach außen trägt.

Nach der Variablendefinition werden in MODUL1.C alle Funktionen aufgeführt, die nicht schon in der Datei MODUL1.H aufgeführt sind. Es kann sich daher nur um die internen, also lokalen Funktionen handeln. Im Prinzip ist diese Definition nicht so wichtig – zumindest bei normalen C-Compilern. ANSI-Compilern bietet sie jedoch die Möglichkeit, sich die Prototypen zu merken und beim Aufruf (hier von „test1") die Parameter zu überprüfen. Die Definition muß sogar erfolgen, auch wenn der Aufruf von „test1" erst auf deren Implementierung folgt. Der Aufruf ist in unserem Beispiel nicht erfolgt, er könnte z.B. weiter unten im Modul noch auftreten.

Die Funktion „test1" ruft nun ihrerseits die Funktion „appl_init" auf. Ihr Prototyp befindet sich in der Datei AES.H, die ja ebenfalls innerhalb von „import.h" eingelesen wird. Danach wird der Variablen „wert" die Konstante 30 (3 * MAX) zugewiesen. Sie behält ihren Wert und wird nicht durch eine Zuweisung an eine Variable mit gleichem Namen in Modul2 verändert. Damit endet Modul1, obwohl natürlich noch mindestens ein Aufruf von „test1" erfolgen sollte, da das Modul sonst keine sinnvolle Aufgabe hätte.

Für Modul2 gilt nun Ähnliches wie für Modul1. In der Datei MODUL2.H wird die Funktion „test2" definiert. Damit ist ihr Prototyp bekannt und sie kann von anderen Modulen aufgerufen werden, wenn diese MODUL2.H importieren. Der Prototyp dient aber auch der Implementierung der Funktion in der Datei MODUL2.C.

In MODUL2.C werden zuerst alle Dateien angegeben, die importiert werden sollen. Das bedeutet, daß nach der Angabe von „import.h" nun die Datei MODUL1.H eingebunden wird. Wir erinnern uns daran, daß GLOBAL auf EXTERN gesetzt ist und somit die importierten Variablen keinen zusätzlichen Speicherplatz benötigen. Durch Einbinden von „modul1.h" werden also die Bezeichner „MAX", „array" und „gl_apid" bekannt. Diese können nun nach Belieben benutzt werden.

Es folgen die Include-Anweisungen „export.h" und „modul2.h". Das bedeutet, daß alle Definitionen von MODUL2.H anderen Modulen bekannt gemacht werden, sofern diese die entsprechende Datei importieren.

Danach werden auch hier alle lokalen Variablen aufgeführt. Die Variable „wert" ist hier nicht gleichbedeutend mit der Variablen desselben Namens in Modul1. Das liegt daran, daß sie mittels LOCAL (= static) definiert wurde.

Nun kann die schon in der Datei MODUL2.H aufgeführte Funktion „test2" implementiert werden. Der Prototyp existiert schon durch Einbinden der Headerdatei (s.o.). Innerhalb der Funktion „test2" werden nun der Parameter „x", die globalen Variablen „gl_apid" und „array" (aus Modul1) sowie die Konstante MAX benutzt. Damit endet auch Modul2. Weitere Funktionen könnten folgen.

Wir nehmen nun an, daß der Typ WORD der Komponenten des Feldes „array" nicht mehr ausreicht und daß dort Langzahlen gespeichert werden sollen. Es genügt nun, in der Datei MODUL1.H die Definition

```
GLOBAL WORD array [MAX];
```

durch

```
GLOBAL LONG array [MAX];
```

zu ersetzen. An der Datei MODUL2.C muß nun keine Änderung vorgenommen werden. Ein erneutes Übersetzen genügt hier vollkommen, es sei denn, man möchte den Typ des Parameters „x" der Funktion „test2" ebenfalls auf LONG erweitern.

So wie unser Beispiel untergliedern wir also auch unsere Applikation in Module, die überschaubar sind, eine abgegrenzte Aufgabe haben und möglichst unabhängig von den anderen Modulen sind. Durch die möglichst große Unabhängigkeit (nur wenige Bezeichner importieren) erreichen wir auch, daß bei Änderung der Schnittstelle eines Moduls nur die betroffenen Module übersetzt werden müssen.

Wir wollen nun erreichen, daß die Module, die wir schreiben, möglichst die gleiche Schnittstelle haben. So können wir uns leichter in die Module hineinversetzen und müssen auch nicht lange überlegen, welche Module wir mit welchen Parametern aufrufen. Diese Grundfunktionen soll jedes Modul zur Verfügung stellen. Sie können jeweils über die Schnittstelle aufgerufen werden.

Jedes Modul hat einen Namen. Dies ist der Name, der auch als Rumpf für den Dateinamen dient. So besteht ein Modul mit Namen WINDOWS aus den beiden Dateien WINDOWS.H und WINDOWS.C. Der Name des Moduls wird für spezielle Funktionsaufrufe benutzt.

Eine oft benötigte Funktion ist das Initialisieren eines Moduls. Meistens muß dies die erste Anweisung sein, die in einem Modul aufgerufen werden muß, bevor weitere Funktionen benutzt werden können. Analog dazu kann man auch fordern, daß ein Modul terminiert werden muß, bevor ein Programm endgültig verlassen wird.

Aus diesem Grund führen wir zwei Funktionen ein, die ein Modul immer haben sollte: das Initialisieren und das Terminieren. Zusammen mit dem Namen des Moduls sollen somit die Funktionen „init_module" und „term_module" zur Verfügung stehen. Für ein Modul mit Namen WINDOWS wären dies also die Funktionen „init_windows" und „term_windows". Meistens kann man auf Parameter verzichten. In unserer Beispielapplikation haben nur zwei Module Parameter bei den Init-Aufrufen. Die Funktionen sind also immer in der entsprechenden Headerdatei zu finden.

Vier weitere „Grundfunktionen" sollen für spezielle Dialogmodule zur Verfügung gestellt werden. Wir sprechen von Dialogmodule, wenn es sich um Module handelt, die Objekte darstellen, die mit dem Benutzer interagieren. Dies umso mehr, wenn es sich um fenstererzeugende Module handelt, d.h. um Module, die in irgendeiner Art Fenster öffnen können. Es kann sich aber auch nur um Module handeln, die eine Art von Piktogramm verwalten, wie Diskette, Drucker oder Klemmbrett.

Für diese Art von Modulen sollten mindestens die folgenden Funktionen zur Verfügung stehen, die für jedes Modul die gleichen Parameter besitzt. Die Funktionen werden hier nur kurz erläutert, eine ausführliche Erklärung mit vielen Beispielen folgt in den weiteren Kapiteln.

– Kreieren eines Fensters

GLOBAL WINDOWP crt_module (...);

– Öffnen eines Fensters (oder einer Dialogbox etc.)

GLOBAL BOOLEAN open_module (...);

– Information eines Fensters

GLOBAL BOOLEAN info_module (...);

– Hilfe eines Fensters (...);

GLOBAL BOOLEAN help_module (...);

Statt „module" muß nun jeweils der Name eingesetzt werden, also z.B. „crt_meta", „open_meta" usw. für das Modul META (Metadatei-Verwaltung).

Somit haben wir sechs Funktionen, die in diesen Modulen immer die gleiche Wirkung haben. Dadurch werden wir später aufs leichteste Menüs implementieren können, die die Funktionen „Öffnen", „Info" und „Hilfe" beinhalten.

Im folgenden soll eine Kurzübersicht über die einzelnen Module gegeben werden, die dann in Abschnitt 6.5 genauestens erklärt werden. Abbildung 6.2 zeigt einen Überblick.

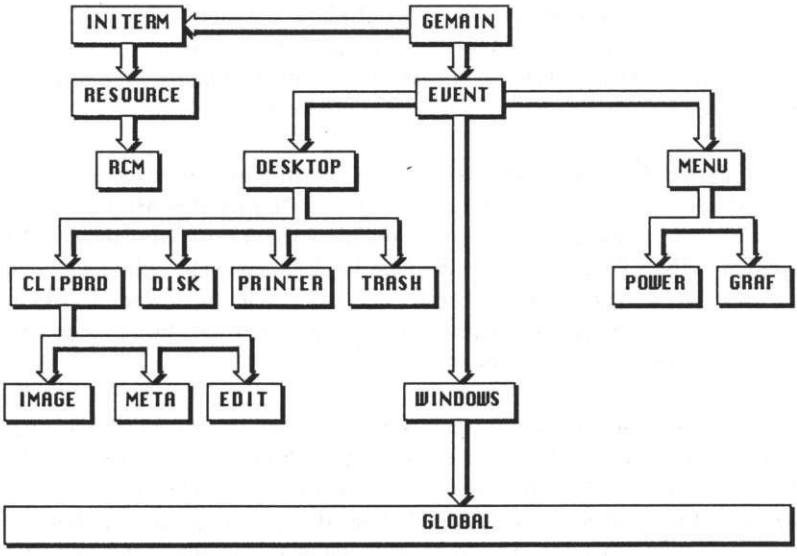



**Abbildung 6.2:** Übersicht über die Module der Applikation

Dabei bedeuten die Pfeile, daß ein Modul eine Funktion eines anderen Moduls aufruft. Der Übersicht halber wurden nicht alle Pfeile eingetragen. Vom Modul INITERM wird jedes Modul mindestens zweimal aufgerufen, einmal zum initialisieren, einmal zum terminieren.

Umgekehrt wird das Modul GLOBAL von fast allen anderen Modulen aufgerufen. Auch das Modul WINDOWS wird außer von EVENT auch von einigen anderen Modulen aufgerufen.

Wir unterscheiden noch zwischen programmabhängigen und -unabhängigen Modulen. Einige Module wurden so konzipiert, daß sie in jeder Applikation verwendet werden können, ohne daß dabei etwas umgeschrieben werden muß, ähnlich wie die Funktionen in der Standard-C-Bibliothek. Zu den programmunabhängigen Modulen gehören GLOBAL, WINDOWS und RCM. In gewissem Sinne zählen dazu aber auch noch GEMAIN und EVENT.

Die Module haben im einzelnen folgende Aufgaben:

– GEMAIN

GEM-unabhängiges Hauptprogramm für alle GEM-Versionen.

– GLOBAL

Globale Vereinbarungen für alle GEM-Applikationen. Dazu zählen immer wiederkehrende Konstanten (z.B. ASCII-Zeichen), Typen (z.B. Zeichenketten, Mengen) und Variablen (z.B. Applikationsnummer, GEM-Arrays, globale Zeichensatz- und Desktop-Größen).

Eine Menge von nützlichen Funktionen sind ebenfalls vorhanden. Mausfunktionen, Objektfunktionen, Dialogfunktionen inklusive Pop-Up-Menüs und Verwaltung von Menütastenkürzeln, Rechteckfunktionen, Speicherfunktionen, Mengenfunktionen, Dateifunktionen.

Dazu bietet das Benutzen dieses Moduls die Garantie für Kompatibilität mit allen GEM-Versionen.

– WINDOWS

GEM-übergeordneter Window-Manager, der Fenster mit verschiedenen Inhalten verwaltet. Neben wichtigen Konstanten und Typen (allgemeiner Fenstertyp) gibt es eine Menge Funktionen zum Verwalten von Fenstern. Fenster-Suchfunktionen, automatische Schieberverwaltung, automatische Objektverwaltung innerhalb eines Fensters, Funktionen zur vereinfachten Verwaltung des Fensterinnern und Menüverwaltung für Menüzeilen in Fenstern.

– RCM

Das Resource-Create-Modul wurde schon in Kapitel 2 angesprochen und dient dazu, Resource-Dateien in C einzubinden, damit diese nicht mehr geladen werden müssen. Interessant ist dies vor allem für Accessories. So hat z.B. das Kontrollfeld keine Resource-Datei.

– EVENT

Der Event-Manager verwaltet alle auftretenden Ereignisse und leitet sie an die anderen Module weiter. Zu den Ereignissen gehören Tastaturereignisse, Mausknopfereignisse, Mausbewegungsereignisse für verschiedene Mausformen, Nachrichtenereignisse (Menüs, Fensternachrichten) und Zeitereignisse für Multitasking.

– INITERM

Alle Module werden von INITERM initialisiert und am Ende terminiert.

– RESOURCE

Resource-Verwaltung mit und ohne explizite Resource-Datei, benutzerdefinierte Objekte, Transformieren von Icons und Bit-Images.

– MENU

Die Menüverwaltung für die globale Menüzeile.

– DESKTOP

Beispiel für einen eigenen Desktop mit einigen Piktogrammen.

– CLIPBRD

Das Modul verwaltet das Klemmbrett. In ihm sind auch Funktionen zu finden, welche die fehlenden Scrap-Funktionen aus GEM/3 im alten GEM nachbildet.

– DISK

Verwaltung des Disk-Piktogramms im Desktop.

– PRINTER

Verwaltung des Drucker-Piktogramms im Desktop.

– TRASH

Verwaltung des Mülleimers, der auch als Fenster geöffnet werden kann.

– IMAGE

Darstellung von Bit-Image-Dateien in einem Fenster.

– META

Darstellung von Metadateien in einem Fenster.

– EDIT

Darstellung von Dateien (ASCII-Textdateien) in einem Fenster.

– POWER

Beispielmodul für die Darstellung von Potenzen einer Zahl inklusive Benutzung von Pop-Up-Menüs in einem Fenster.

– GRAF

Beispielmodul für die Darstellung einer Grafik inklusive Multitasking-Benutzung.

Die Aufrufreihenfolge der Module ist von großer Wichtigkeit. Ein Modul muß zunächst initialisiert werden, bevor man seine Funktionen aufrufen kann.

Beim Start eines Programms wird zunächst GEMAIN aufgerufen. Dort wird das Modul INITERM angesprungen und alle Module initialisiert. Dann wird ins EVENT-Modul verzweigt und auf Ereignisse gewartet. Tritt ein Menüereignis auf, so wird ins Modul MENU verzweigt, und von dort werden die entsprechenden Funktionen aufgerufen, wie z.B. „open_graf", um ein Grafikfenster zu öffnen.

Bei Fensterereignissen wird über WINDOWS in das jeweilige Modul verzweigt. Falls dies der Desktop ist, wird von ihm die entsprechende Icon-Funktion für CLIPBRD, DISK, PRINTER oder TRASH aufgerufen. Über CLIPBRD können wiederum Bit-Image-Dateien, Metadateien und Textdateien aufgerufen werden.

## 6.3 Allgemeine Fensterverwaltung

Für die Verwaltung von Fenstern in einer grafischen Benutzeroberfläche muß immer ein größerer Aufwand getrieben werden, als für die Verwaltung von einfachen Dialogen. Der Grund ist darin zu suchen, daß ein Fenster, einmal geöffnet, solange lebt, bis es wieder geschlossen wird. Da dies für mehrere Fenster gleichzeitig gilt, müssen alle Informationen für die Fenster immer zugriffsbereit sein. Ein Fenster sieht dann wie ein Prozeß aus, der seine eigenen Informationen verwaltet, die von den anderen Fenstern unabhängig sind. Da Fenster in der Regel verschiedene Informationen anzeigen, müssen Algorithmen

angewandt werden, die abhängig vom jeweiligen Fenster den aktuellen Stand der Information anzeigen.

Bevor wir näher auf die Verwaltung verschiedener Fensterinhalte eingehen, wollen wir uns zunächst ein Fenster genauer betrachten. Abbildung 6.3 zeigt ein Fenster des Resource-Construction-Sets von Digital Research.

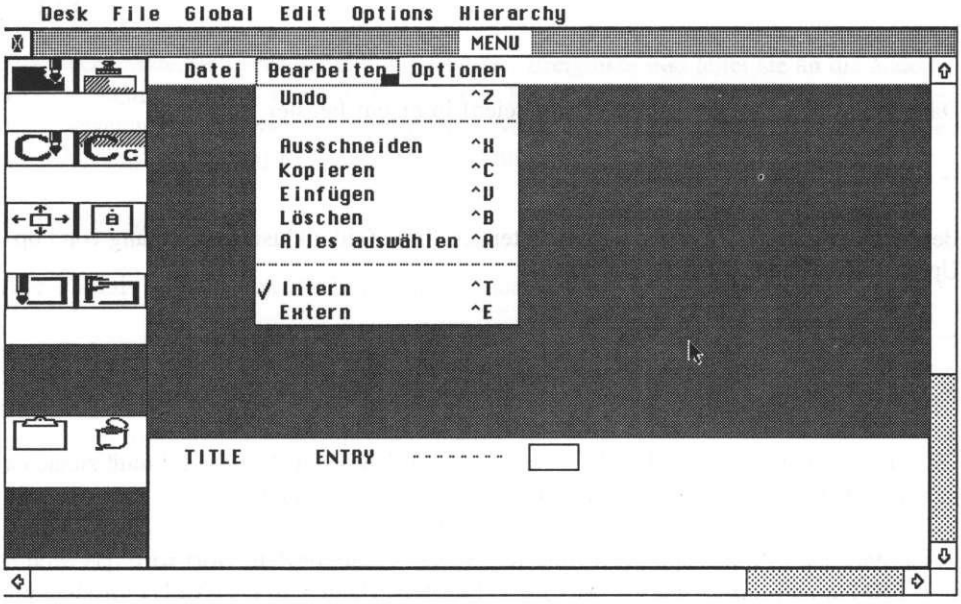



**Abbildung 6.3:** Fenster im Resource-Construction-Set.

An diesem Fenster werden Methoden benutzt, die auf den ersten Blick nicht auffallen. Das Fenster zeigt am linken Rand eine Leiste mit Tools, die mit dem Befehl „Hide Tools" versteckt werden kann. Am unteren Rand wird eine Leiste mit Teilen gezeigt, die mit dem Befehl „Hide Parts" versteckt werden kann.

Der Rest des Fensters ist der Bereich, der die eigentliche Information anzeigt. Dies ist auch der Bereich, der gescrollt werden kann und der für die Berechnung der Schieber relevant ist. Wenn das Fenster gescrollt wird, so bleiben der linke und untere Rand also erhalten. Man könnte sich außerdem vorstellen, einen oberen und einen rechten Rand zu haben. Am oberen Rand könnten dann z.B. eine Menüzeile dargestellt werden und am rechten Rand weitere Bedienungselemente, wie das z.B. im Programm GEMPAINT von Digital Research der Fall ist.

Da der Programmierer für das gesamte Innere des Fensters verantwortlich ist, der Bereich, der gescrollt werden kann, jedoch manchmal auch kleiner ausfällt, wollen wir das Fensterinnere noch einmal unterteilen. In Abbildung 6.4 wird ein Fenster gezeigt, welches sämtliche Komponenten enthält.

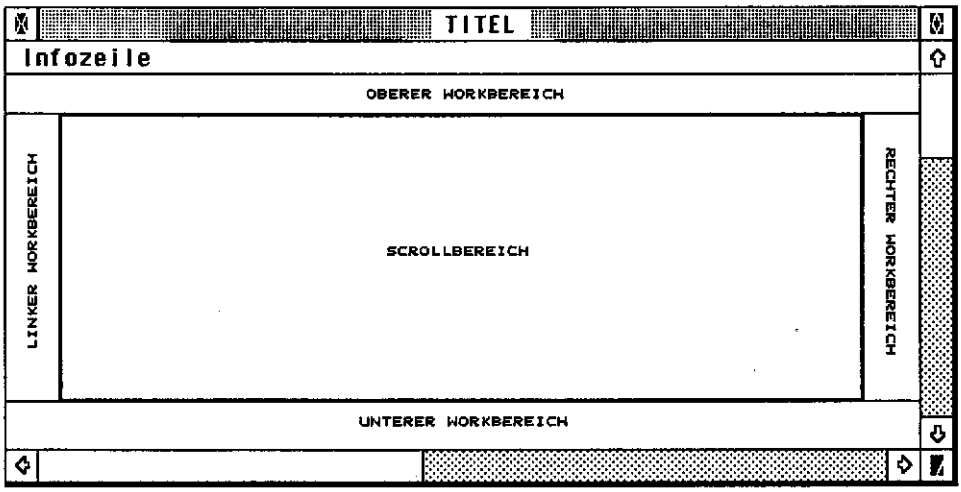



**Abbildung 6.4:** Allgemeiner Aufbau eines Fensters

Wir erkennen folgendes: Zunächst existiert der Fensterrandbereich. Welche Komponenten er erhält, muß beim Erzeugen des Fensters angegeben werden. Das Fensterinnere nennen wir den Arbeitsbereich (Workbereich). Für diesen ist der Programmierer verantwortlich. Innerhalb des Arbeitsbereichs liegt die eigentliche Information, d.h. der Teil eines Dokuments, durch den gescrollt werden kann und für den die Schieber die relative Größe anzeigen. Wir nennen ihn deshalb auch Scrollbereich.

Der Scrollbereich muß nicht kleiner sein als der Arbeitsbereich. Falls keine zusätzlichen Komponenten wie in Abbildung 6.3 benötigt werden, entfällt die Unterscheidung zwischen Arbeitsbereich und Scrollbereich – sie sind identisch.

Man hat nun die Möglichkeit, den oberen Rand (top), den linken Rand (left), den unteren Rand (bottom) und den rechten Rand (right) zu belegen (oder nur einen Teil der Ränder). Beim Scrollen werden dann die Ränder berücksichtigt und nicht mitgescrollt. Sie bleiben ergo erhalten.

Das hier beschriebene Modul WINDOWS berücksichtigt eine Menge von Formen von Fenstern, die schon in verschiedenen bekannten Programmen aufgetreten sind, und versucht, einfache Hilfsmittel zur Verfügung zu stellen, um alle diese Formen zu unterstützen.

Dafür wird eine Struktur zur Verfügung gestellt, die mit einem Fenster verbunden ist. Diese Struktur ist in der Datei WINDOWS.H definiert und hat den Namen WINDOW. Ein Zeiger (Pointer) auf eine solche Struktur ist (als WINDOWP) definiert.

Diese Struktur sollte man verstanden haben, da sie als Parameter an fast alle Funktionen des Window-Managers übergeben wird. Bevor die Struktur genauer erläutert wird, muß

aber noch ein Konzept vorgestellt werden, welches für das Verstehen unabdingbar ist. Es handelt sich um das Konzept der indirekten Funktionen, welches nur in modernen höheren Programmiersprachen verfügbar ist. So kann man diese in C oder Modula-2 definieren. Sprachen wie Basic sind dabei selbstverständlich ausgeklammert.

Fast jeder von uns kennt den Aufruf einer Funktion von irgendeiner Stelle eines Programms aus. Beispiel:

```
#include <portab.h>
#include <math.h>

GLOBAL DOUBLE berechne (x, y)
DOUBLE x, y;

{
  DOUBLE res, d;

  d = sin (x);
  /* Benutzung von d für weitere Berechnungen */
  d = 3 * sin (y + 1.1);
  /* Benutzung von d für weitere Berechnungen */
  /* ... */
  return (res);
}
```

Ein Aufruf von „berechne" könnte dann als

```
berechne (3.0, 5.1);
```

geschrieben werden.

In der Funktion „berechne" wird die Funktion „sinus" aufgerufen, deren Definition sich in der Datei „math.h" befindet. Dies ist eine Möglichkeit, die Sinusfunktion aufzurufen. Sie wird hier zweimal aufgerufen, man könnte sich auch noch weitere Aufrufe vorstellen. Dieselben Berechnungen sollen jetzt für den Cosinus gemacht werden. Dabei müßte man überall, wo „sin" steht, das Wort „cos" einsetzen. Die Funktion kann dann aber nicht mehr für die Berechnung über den Sinus verwendet werden.

Eine Möglichkeit besteht nun darin, an die Funktion „berechne" eine Zahl mitzuliefern, welche die Funktion angibt und innerhalb von „berechne" mit Hilfe von „switch" entsprechend zu verzweigen.

Die Implementierung würde dann z.B. aussehen:

```
GLOBAL DOUBLE berechne (func, x, y)
WORD   func;
DOUBLE x, y;

{
  DOUBLE res, d;

  switch (func)
  {
    case 0 : d = sin (x);
    case 1 : d = cos (x);
  }
  /* Benutzung von d für weitere Berechnungen */
  switch (func)
  {
    case 0 : d = 3 * sin (y + 1.1);
    case 1 : d = 3 * cos (y + 1.1);
  }
  /* Benutzung von d für weitere Berechnungen */
  /* ... */
  return (res);
}
```

Ein Aufruf lautet dann

berechne (0, 3.0, 5.1)

für den Sinus und

berechne (1, 3.0, 5.1)

für den Cosinus.

Wir sehen ganz deutlich die Schwäche dieser Implementierung. Für jede Funktion muß in einem „switch"-Befehl ein Fall vorgesehen werden. Noch schlimmer: Da die Funktion einige Male benutzt wird, müssen es jeweils mehrere „switch"-Implementierungen sein. Das Problem bei dieser Implementierung ist aber nicht nur die rasch wachsende Codegröße der Funktion „berechne", sondern auch das Fehlen weiterer Funktionen. Für die gleiche Berechnung mit Hilfe des Tangens ist die Funktion so nicht vorgesehen. Wünschenswert wäre eine Implementierung, die es erlaubt, innerhalb von „berechne" die Funktion aufzurufen, die gerade benötigt wird, ohne daß dort die Funktion zur Übersetzungszeit selbst bekannt ist.

Dies geht nur mit dem Prinzip des indirekten Funktionsaufrufs. Die Funktion, die ausgeführt werden soll, wird dann als Parameter an „berechne" übergeben. Die Definition sieht nun wieder einfacher aus:

```
GLOBAL DOUBLE berechne (func, x, y)
DOUBLE (*func) _((DOUBLE x));
DOUBLE x, y;

{
  DOUBLE res, d;

  d = (*func) (x);
  /* Benutzung von d für weitere Berechnungen */
  d = 3 * (*func) (y + 1.1);
  /* Benutzung von d für weitere Berechnungen */
  /* ... */
  return (res);
}
```

Ein Aufruf von „berechne" sieht nun folgendermaßen aus:

berechne (sin, 3.0, 5.1);

oder

berechne (cos, 3.0, 5.1)

Es hindert uns allerdings auch niemand daran, die Funktion „berechne" als

berechne (tan, 3.0, 5.1);

oder

berechne (atan, 3.0, 5.1);

aufzurufen. Lediglich ein Aufruf von z.B.

berechne (fmod, 3.0, 5.1);

würde nicht funktionieren. Die Modulo-Funktion „fmod" erwartet nämlich zwei Parameter, nicht nur einen wie etwa Sinus und Cosinus. Da die Funktion „func" in „berechne" aber nur jeweils mit einem Parameter aufgerufen wird, würde an „fmod" nur ein Parameter übergeben werden, was in diesem Fall falsch ist. Der Aufruf indirekter Funktionen (func) klappt also nur, wenn die Anzahl, die Reihenfolge und der Typ der Parameter über-

einstimmt. Ähnliches gilt für das Funktionsergebnis. Bei einem ANSI-Compiler würde der letzte „berechne"-Aufruf nicht angenommen werden. Es würde eine Fehlermeldung geliefert und erst gar kein Code erzeugt werden. Bei anderen Compilern würde das Programm zur Laufzeit merkwürdige Ergebnisse bringen. Das ist ein Grund mehr, einen ANSI-C-Compiler zu benutzen.

Wir haben nun gesehen, wie man mit Hilfe indirekter Funktionsaufrufe eine Routine schreiben kann, die noch nicht weiß, welche Funktion letztendlich aufgerufen wird. Diese wird durch den Übergabeparameter „func" beschrieben.

Genauso kann man aber auch Variablen definieren, die Zeiger auf Funktionen enthalten und diese Variablen statt der eigentlichen Funktion aufrufen. Insbesondere können mehrere dieser Funktionen (Variablen) in einer Struktur zusammengefaßt werden. In folgendem Beispiel sollen viele Funktionen an eine Routine übergeben werden. Da man die Anzahl der Parameter aber gering halten möchte, werden die Funktionen erst einmal in einer Struktur gespeichert und von der ausführenden Routine dann aufgerufen.

```
#include <portab.h>
#include <math.h>


typedef struct
{
  DOUBLE (*func1) _((DOUBLE x));
  DOUBLE (*func2) _((DOUBLE x));
  DOUBLE (*func3) _((DOUBLE x));
  DOUBLE (*func4) _((DOUBLE x, DOUBLE y));
} FUNCS;


GLOBAL DOUBLE berechne (funcs, x, y)
FUNCS  *funcs;
DOUBLE x, y;


{
  DOUBLE res, d;


  d = (*funcs->func1) (x);
  /* Benutzung von d für weitere Berechnungen */
  d = 3 * (*funcs->func4) (x, y + 1.1);
  /* Benutzung von d für weitere Berechnungen */
  /* ... */
  return (res);
}
```

Zunächst muß nun eine Struktur vorliegen, welche initialisiert wird. Diese wird dann an die Funktion „berechne" übergeben. Also:

```
{
  FUNCS funcs;

  funcs.func1 = sin;
  funcs.func2 = cos;
  funcs.func3 = tan;
  funcs.func4 = fmod;

  berechne (&funcs, 13.7, 5.91);
}
```

In diesem Beispiel werden den verschiedenen Variablen innerhalb der Struktur „funcs" zuerst Werte für die Funktionen zugewiesen. Man beachte, daß für func4 eine Funktion gewählt werden muß, welche zwei Parameter erwartet, also z.B. „fmod". Ansonsten ist das Prinzip das gleiche wie oben. Der Vorteil liegt in der besseren Übersicht der Funktion „berechne", da diese nun immer nur drei Parameter zu haben braucht, auch wenn noch weitere Funktionen in der Struktur FUNCS dazukommen würden.

Wir fassen zusammen: Eine Funktion muß nicht notwendigerweise direkt aufgerufen werden. Ein indirekter Aufruf (über den Zeiger) einer Funktion hat den Vorteil, unabhängig von der eigentlichen Funktion zu sein, die beim Aufruf dahintersteckt. Routinen, die allgemein geschrieben werden müssen, aber mit verschiedenen Funktionen auf die gleiche Art zu arbeiten haben, können mit dem Prinzip des indirekten Aufrufs implementiert werden.

Beispiele für Aufrufe mit indirekten Funktionen sind auch in vielen Compiler-Bibliotheken enthalten. So existiert z.B. eine Funktion „qsort", welche den Quicksort-Algorithmus auf ein Feld von Elementen anwendet. Da aber die Komponenten des Feldes verschiedener Art sein können (Ganzzahlen, Fließkommazahlen, Zeichenketten etc), kann die Funktion „qsort" nicht den Vergleich von zwei Komponenten durchführen. Sie benötigt dann einen Zeiger auf eine Funktion, die diesen Vergleich durchführt. Der Aufrufer kann nun je nach Art der Elemente verschiedene Funktionen an „qsort" übergeben, z.B. „strcmp", um Zeichenketten zu vergleichen oder „longcmp", um lange Ganzzahlen zu vergleichen.

Doch nun zurück zu unserer Struktur WINDOW in WINDOWS.H. Sie besteht aus 41 Variablen, 21 gewöhnlichen Variablen und 20 Zeigern auf Funktionen. Jede Variable oder Funktion hat eine wichtige Bedeutung, die nun beschrieben werden soll.

```
typedef struct window
{
  WORD    handle;
  WORD    opened;
  UWORD   flags;
  UWORD   kind;
  WORD    class;
  WORD    icon;
  LRECT   doc;
  WORD    xfac;
  WORD    yfac;
  RECT    scroll;
  RECT    work;
  WORD    mousenum;
  MFORM   *mouseform;
  LONG    milli;
  LONG    count;
  LONG    special;
  STRING  name;
  STRING  info;
  OBJECT  *object;
  OBJECT  *menu;
  WORD    first_menu;
  VOID    (*updt_menu) _((WINDOWP window));
  VOID    (*hndl_menu) _((WINDOWP window, WORD title, WORD item));
  BOOLEAN (*test)      _((WINDOWP window, WORD action));
  VOID    (*open)      _((WINDOWP window));
  VOID    (*close)     _((WINDOWP window));
  VOID    (*delete)    _((WINDOWP window));
  VOID    (*draw)      _((WINDOWP window));
  VOID    (*arrow)     _((WINDOWP window, WORD dir,
                          LONG oldpos, LONG newpos));
  VOID    (*snap)      _((WINDOWP window, RECT *new, WORD mode));
  VOID    (*objop)     _((WINDOWP window, SET objs, WORD action));
  WORD    (*drag)      _((WINDOWP src_window, WORD src_obj,
                          WINDOWP dest_window, WORD dest_obj));
  VOID    (*click)     _((WINDOWP window, MKINFO *mk));
  VOID    (*unclick)   _((WINDOWP window));
  BOOLEAN (*key)       _((WINDOWP window, MKINFO *mk));
  VOID    (*timer)     _((WINDOWP window));
  VOID    (*top)       _((WINDOWP window));
  VOID    (*untop)     _((WINDOWP window));
  VOID    (*edit)      _((WINDOWP window, WORD action));
  BOOLEAN (*showinfo)  _((WINDOWP window, WORD icon));
  BOOLEAN (*showhelp)  _((WINDOWP window, WORD icon));
} WINDOW;
```

WORD handle;

Das GEM-Handle für ein Fenster. Es wird vor allem für die Aufrufe der AES-Window-Bibliothek verwendet. Wenn ein Fenster nicht offen ist, so hat „handle" den Wert −1.

WORD opened;

Gibt an, wie oft ein Fenster geöffnet wurde. Das Fenster muß dann genauso oft geschlossen werden, wie es geöffnet wurde. Beim Anwählen der Schließbox eines Fensters muß das Fenster nicht notwendigerweise wirklich geschlossen werden. Es kann auch sein, daß man in einer Hierarchie nur um eine Stufe zurückfällt. Bestes Beipiel ist der GEM-Desktop. Wird in einem geöffneten Fenster ein Ordner angewählt, so öffnet sich dieser und das Fenster zeigt den neuen Inhalt. Dabei wird aber kein wirklich neues Fenster geöffnet, wie das z.B. beim Finder des Macintosh der Fall ist. Stattdessen wird das alte Fenster benutzt. Wird nun auf die Schließbox geklickt, so soll das Fenster nicht geschlossen werden, da man nur in der Ordnerhierarchie um eine Stufe zurückgeht.

UWORD flags;

Dies sind verschiedene Flags, welche die Applikation beim Kreieren eines Fensters setzen kann, um das Verhalten des Window-Managers zu beeinflussen. Die Flags können sein (siehe WINDOWS.H):

WI_NONE: Keine Flags
WI_FULLED: Das Fenster ist auf voller Größe.
WI_ONTOP: Das Fenster befindet sich ganz oben.
WI_RESIDENT: Position und Größe sollen beim Schließen des Fensters erhalten bleiben. Das bedeutet, daß beim erneuten Öffnen des Fensters dieses an der alten Stelle mit der alten Größe auftaucht. Dieses wird besonders gern benutzt, wenn Fenster beim Schließen in Piktogramme „verwandelt" werden und durch erneutes Anklicken der Piktogramme wieder geöffnet werden.
WI_MOUSE: Das Fenster hat eine eigene Mausform. Der Window-Manager kann zusammen mit dem Event-Manager für jedes Fenster eine eigene Mausform verwalten. Beim Eintreten in den Scrollbereich wird die Mausform umgeschaltet.
WI_DIALOG: Das Fenster zeigt eine „modeless" Dialogbox. Dies ist in dieser Version noch nicht implementiert, jedoch schon vorgesehen.

UWORD kind;

Dies ist die Art des Fensters, die festlegt, welche Komponenten ein Fenster haben kann. Die Komponeneten sind (siehe AES.H) Titelzeile, Infozeile, Schließbox, Fullbox, Vergrößerungsbox und alle Schieberelemente.

WORD class;

Darunter verstehen wird die Klasse eines Fensters. Es handelt sich hierbei um eine einzigartige Identifikation für die Beschreibung des Fensters. Dabei gehören Fenster, die gleichartige Inhalte zeigen, auch in die gleiche Klasse. Beispielsweise wären alle Fenster im Programm Wordplus in der gleichen Klasse, da sie alle Text einer Datei zeigen. Dagegen wären die Ausgabefenster und das DB-Info-Fenster von ADIMENS ST in verschiedenen Klassen, da sie gänzlich verschiedenartige Inhalte zeigen. Bestimmte Konstanten für Klassenidentifikation sind schon reserviert. Es sind dies (siehe GLOBAL.H)

```
#define DESK          0
#define DESKWINDOW    1
```

DESK gibt das Fenster an, welches den Desktop beschreibt, der den ganzen Bildschirm ausfüllt. DESKWINDOW gibt ebenfalls den Desktop an, der sich aber nun in einem Fenster befindet (z.B. für Accessories oder Multitasking X/GEM).

Alle Zahlen ab 2 aufwärts können nun für eigene verschiedenartige Klassen von Fenstern benutzt werden. Beispiele finden sich in allen Modulen von SCRAP, die Fenster erzeugen, also DESKTOP, CLIPBRD, TRASH, IMAGE, META, EDIT, POWER, GRAF.

WORD icon;

Es handelt sich um die Objektnummer des Piktogramms, aus welchem ein Fenster entstanden ist. Falls das Fenster nicht aus einem Piktogramm entstanden ist, kann der Wert auch NIL (= -1) annehmen. Diese Variable wird benutzt, um beim Anklicken eines Piktogramms das dazugehörige Fenster zu suchen, falls bereits eines existiert. Beispiele sind in SCRAP in allen Modulen zu finden, die Fenster durch Anklicken von Piktogrammen öffnen. Weitere bekannte Beispiele für eine sinnvolle Anwendung dieser Variablen sind der GEM-Desktop und das Textverarbeitungsprogramm TEMPUS.

LRECT doc;

Der Datentyp LRECT ist ein Rechteck aus den vier Langzahlen x, y, w und h für die X-Koordinate, die Y-Koordinate, die Breite und die Höhe. Somit existieren die vier Variablen doc.x, doc.y, doc.w und doc.h. Die beiden ersten Komponenten geben die absolute X- und Y-Koordinate innerhalb des Dokumentes an, welches in der linken oberen Ecke des Fensters gezeigt wird. Die beiden anderen Komponenten geben die aktuelle Breite und Höhe des Dokumentes an, von welchem das Fenster einen Ausschnitt zeigt. Mit Hilfe dieser Werte kann der Window-Manager jederzeit die aktuellen Schieberpositionen und -größen berechnen.

Zur Verdeutlichung dient die Abbildung 6.5.

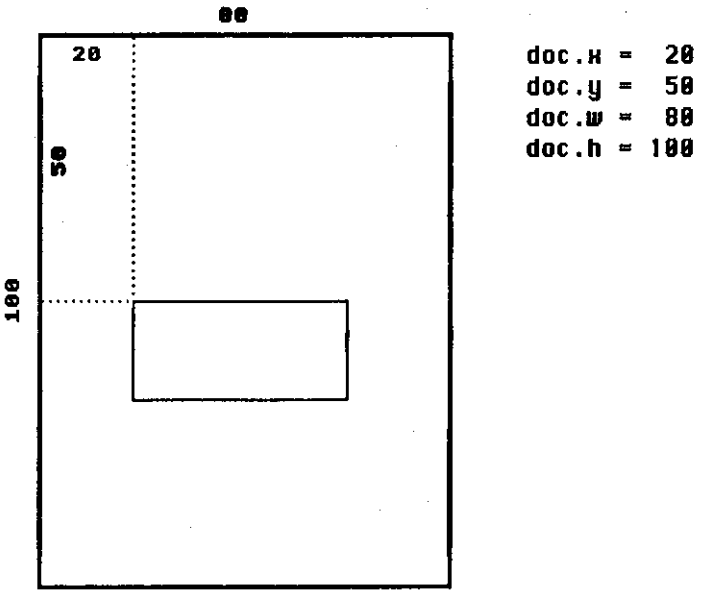



**Abbildung 6.5:** Dokument und Fenster

Jedes Fenster zeigt einen Ausschnitt aus einem (meist größeren) Dokument. Zeigt das Fenster in der linken oberen Ecke beispielsweise die 50. Zeile und 20. Spalte eines Dokumentes an, welches 100 Zeilen mit jeweils 80 Spalten besitzt, so sind die Werte von „doc":

```
doc.x = 20;
doc.y = 50;
doc.w = 80;
doc.h = 100;
```

Die Zählung beginnt dabei jeweils von 0. Sind alle Werte 0, so muß das Fenster leer sein, da noch kein Dokument existiert.

WORD xfac;

Der X-Faktor gibt an, mit welcher Zahl multipliziert werden muß, wenn in horizontaler Richtung um eine Einheit gescrollt werden soll. Eine Einheit ist minimal 1 Pixel. Bei vielen Programmen bieten sich aber größere Zahlen an. Wird beispielsweise in einem Fenster Text dargestellt, so sollte „xfac" z.B. den Wert 8 besitzen, wenn ein Zeichen 8 Pixel

Breite besitzt. Ist ein Zeichen größer oder kleiner, kann der Wert natürlich entsprechend angepaßt werden.

Die Komponenten „doc.x" und „doc.w" von „doc" (s.o.) sind Werte in dieser Einheit. Es ist also für „doc" unwesentlich, mit welcher Schriftart der Text dargestellt wird, maßgebend sind die Anzahl der Zeilen und Spalten eines Dokumentes in Einheiten.

WORD yfac;

Der Y-Faktor gibt an, mit welcher Zahl multipliziert werden muß, wenn in vertikaler Richtung um eine Einheit gescrollt werden soll. Eine Einheit ist minimal 1 Pixel. Bei vielen Programmen bieten sich aber größere Zahlen an. Wird beispielsweise in einem Fenster Text dargestellt, so sollte „yfac" z.B. den Wert 16 besitzen, wenn ein Zeichen 16 Pixel Höhe besitzt. Ist ein Zeichen größer oder kleiner, kann der Wert natürlich entsprechend angepaßt werden.

Die Komponenten „doc.y" und „doc.h" von „doc" (s.o.) sind Werte in dieser Einheit. Es ist also für „doc" unwesentlich, mit welcher Schriftart der Text dargestellt wird, maßgebend sind die Anzahl der Zeilen und Spalten eines Dokuments in Einheiten.

Werden in einem Fenster Piktogramme dargestellt (siehe CLIPBRD), so sind xfac und yfac die Breite und Höhe dieser Piktogramme in Pixel.

RECT scroll;

Der Datentyp RECT ist ein Rechteck aus den vier Ganzzahlen x, y, w und h für die X-Koordinate, die Y-Koordinate, die Breite und die Höhe. Somit existieren die vier Variablen scroll.x, scroll.y, scroll.w und scroll.h. Sie geben die absolute X- und Y-Koordinate sowie Breite und Höhe (in Pixeln) des Scrollbereichs des Fensters an (siehe auch Abb. 6.4). Mit Hilfe dieser Werte (und „doc") kann der Window-Manager jederzeit die aktuellen Schiebergrößen berechnen.

RECT work;

Die Bedeutung ist wie „scroll", nur ist damit der Arbeitsbereich des Fensters gemeint, d.h. der Bereich, für den der Programmierer verantwortlich ist (siehe auch Abb. 6.4).

WORD mousenum;

Jedes Fenster kann seine eigene Mausform haben. Die Nummer der Mausform entspricht den Konstanten in AES.H.

MFORM *mouseform;

Hat „mousenum" den Wert USERDEF (= 255), so muß „mouseform" die Adresse der Mausform enthalten. Sie ist ein Zeiger auf eine Struktur namens MFORM (siehe AES.H). Ein Beispiel hierfür ist im Modul POWER gegeben.

LONG milli;

Dies ist die Anzahl der Millisekunden, die verstreichen sollen, bevor das Fenster eine Aktion durchführen darf. Ist der Zähler 0, so handelt es sich um kein Multitaskingfenster. Ein Beispiel wird im Modul GRAF gegeben. Die Genauigkeit hängt davon ab, welcher Wert für den Timer beim Aufruf von „evnt_multi" übergeben wird (siehe Modul EVENT). Soll z.B. der Inhalt eines Fensters alle 10 Minuten auf Diskette gespeichert werden, so muß in „milli" der Wert 10 * 60 * 1000 = 600000 eingetragen werden.

LONG count;

Der Zähler für die Millisekunden wird intern benutzt. Immer wenn „count" größer ist als „milli", wird ein Zeitereignis für das Fenster ausgelöst.

LONG special;

Es handelt sich um einen speziellen Wert für jedes Fenster. Der Wert ist von der Klasse des Fensters abhängig und kann nicht nur eine Langzahl speichern, sondern vielmehr einen Zeiger auf eine Struktur. Die Struktur kann praktisch beliebige Größe haben. Damit können zu jedem Fenster beliebige Informationen gespeichert werden. Beispielsweise kann darin ein Zeiger auf den Text gespeichert werden, der in einem Fenster dargestellt wird.

Alle Variablen, die in einer Struktur gespeichert werden, auf die „special" zeigt, sind lokal zu diesem Fenster. Das bedeutet, daß, wann immer eine Operation auf diesem Fenster geschieht, man auf diese Variablen zugreifen kann. Alle Fenster leben damit unabhängig voneinander. Dies ist auch die Voraussetzung für ein sinnvolles Multitasking der Fenster innerhalb eines Programms. Jedes Fenster entspricht damit einer Task, die auf ihren eigenen lokalen Variablen operiert. Beispiele für die Benutzung einer Struktur in der Variablen „special" finden wir im Modul EDIT.

STRING name;

Der Name des Fensters wird hier abgelegt.

STRING info;

Die Infozeile des Fensters wird hier abgelegt.

OBJECT *object;

Dies stellt einen Zeiger auf eine Objektbaumstruktur, die im Scrollbereich des Fensters angezeigt werden soll, dar. Ist er NULL, so wird kein Objektbaum im Fenster erwartet. Falls ein Objektbaum im Fenster angemeldet wird, so übernimmt der Window-Manager dessen Verwaltung automatisch. Das bedeutet, daß sich die Applikation um folgende Dinge nicht kümmern muß: Zeichnen des Fensterinhaltes, Scrolling des Fensterinhalts, Setzen der Schieber und Einrasten des Fensters auf bestimmte Positionen.

OBJECT *menu;

Damit meinen wir einen Zeiger auf eine Objektbaumstruktur, die eine Menüzeile darstellt. Ist er NULL, so wird keine Menüzeile im Fenster erwartet. Falls eine Menüzeile im Fenster angemeldet wird, übernimmt der Window-Manager deren Verwaltung automatisch. Die Applikation muß lediglich noch dem Window-Manager bekannt machen, welche Routine aufgerufen werden muß, um die Menüs auszuführen und um sie auf den neuesten Stand zu bringen.

WORD first_menu;

Dies ist die Nummer des ersten im Fenster angezeigten Menüs. Der Wert wird nur intern für den Window-Manager benötigt. Falls die Menüzeile nicht ganz ins Fenster paßt, kann sie gescrollt werden, so daß die nicht sichtbaren Teile ebenfalls angewählt werden können.

Nun folgen die 20 Zeiger auf die Funktionen, die bei bestimmten Fensteroperationen ausgelöst werden sollen. Das Prinzip ist ungefähr folgendes: Bei einer Fensteraktion (z.B. beim Öffnen eines Fensters) wird vom Window-Manager nachgesehen, ob es eine Funktion gibt, die ausgeführt werden soll. Ist das der Fall (der Zeiger also nicht NULL), so wird sie angesprungen. Der indirekte Sprung zu Funktionen wurde bereits weiter oben erklärt. Da der Window-Manager die eigentliche Funktion nicht kennt, die dahintersteckt, kann das Fenster beim Initialisieren jede beliebige Routine angeben, sofern deren Parameter übereinstimmen. So kann zum Beispiel beim Öffnen eines Fensters eine sich vergrößernde Box gezeichnet werden oder ein Zugriffspfad für eine Datenbank initialisiert werden.

Bei den folgenden Erklärungen werden die Parameter jeweils in ANSI-C-Schreibweise angegeben. Der Parameter „window“ ist in allen Fällen ein Zeiger auf die aktuelle Window-Struktur des Fensters.

VOID (*updt_menu) (WINDOWP window);

Die Funktion wird vom Window-Manager aufgerufen, bevor die Menüzeile gezeichnet wird, wenn der Benutzer die Menüzeile angeklickt hat oder wenn er eine Taste drückt,

für die ein Menüpunkt definiert wurde. Insbesondere sollte die Funktion genau dann initialisiert werden, wenn eine Menüzeile für ein Fenster definiert wurde. In dieser Funktion werden üblicherweise die Menüs eingestellt, d.h. einzelne Menütitel oder Menüeinträge grau oder abgehakt eingestellt (DISABLED oder CHECKED). Beispiele sind in den Modulen CLIPBRD und EDIT zu finden.

VOID (*hndl_menu) (WINDOWP window, WORD title, WORD item);

Die Funktion wird vom Window-Manager aufgerufen, wenn der Benutzer ein Menü in einem Fenster angewählt hat. „title" ist die Nummer des Titels, „item" die Nummer des Eintrags im Menübaum. Diese Funktion sollte initialisiert werden, wenn eine Menüzeile für ein Fenster definiert wurde. Ein Beispiel ist in dem Modul CLIPBRD zu finden.

BOOLEAN (*test) (WINDOWP window, WORD action);

Diese Funktion wird vom Window-Manager aufgerufen, bevor ein Fenster geschlossen oder gelöscht wird. Der Parameter „action" gibt dabei die Aktion an, die gerade ausgeführt werden soll. Weiterhin wird die Funktion im Modul MENU aufgerufen, um zu testen, ob Cut/Copy/Paste-Operationen in diesem Fenster möglich sind. Der Funktionswert ist TRUE, wenn das Fenster geschlossen oder gelöscht werden darf oder wenn Cut/Copy/Paste-Operationen möglich sind. Die Werte von „action" können sein (siehe WINDOWS.H):

DO_UNDO: Test, ob eine Undo-Operation möglich ist. Das ist üblicherweise nach dem Verändern eines Dokuments der Fall.
DO_CUT: Test, ob eine Cut-Operation möglich ist. Das ist üblicherweise nach Markieren eines Blocks der Fall.
DO_COPY: Test, ob eine Copy-Operation möglich ist. Das ist üblicherweise nach Markieren eines Blocks der Fall.
DO_PASTE: Test, ob eine Paste-Operation möglich ist. Das ist üblicherweise nach einer Cut- oder Copy-Operation der Fall, wenn also der Puffer gefüllt ist.
DO_CLEAR: Test, ob eine Clear-Operation möglich ist. Das ist üblicherweise nach Markieren eines Blocks der Fall.
DO_SELALL: Test, ob eine Select-All-Operation möglich ist. Dies sollte immer der Fall sein.
DO_CLOSE: Test, ob das Fenster nach Anklicken der Schließbox wirklich geschlossen werden soll. An dieser Stelle kann z.B. eine Abfrage erfolgen, ob das Dokument, welches im Fenster dargestellt wird, noch auf die Diskette gerettet werden soll. Wählt der Benutzer den Abbruch-Knopf, so wird das Fenster doch nicht geschlossen.
DO_DELETE: Test, ob das Fenster gelöscht werden soll. An dieser Stelle kann wie oben eine Sicherheitsabfrage erfolgen. Dies ist insbesondere dann sinnvoll, wenn beim normalen Schließen der Fensterinhalt nicht gelöscht wird, d.h. das Flag WI_RESIDENT (s.o.) eingeschaltet ist. Ein Beispiel finden wir beim Texteditor

TEMPUS. Dort kann das Fenster durch Anklicken des entsprechenden Icons wieder geöffnet werden. Die Sicherheitsabfrage müßte dann z.B. bei der Operation „Text-Icon auf Papierkorb-Icon ziehen" erfolgen.
DO_EXTERNAL: Ist zusätzlich das niederwertige Bit im oberen Byte gesetzt, dann sollen sich die obigen Operationen (Undo, Cut, Copy, Paste) auf den externen Puffer beziehen, d.h. auf das GEM-Clipboard. Bei einer Abfrage muß also immer das obere Byte mit ausgewertet bzw. ausgeblendet werden.

Beispiele finden sich in den Modulen MENU und EDIT.

VOID (*open) (WINDOWP window);

Die Funktion wird unmittelbar vor dem Öffnen eines Fensters aufgerufen. Sinnvolle Aktionen an dieser Stelle können sein: Zeichnen einer sich ausdehnenden Box oder Initialisieren eines Zugriffspfads etc. Beispiele sind in allen Modulen angegeben, welche Fenster benutzen.

VOID (*close) (WINDOWP window);

Die Funktion wird unmittelbar nach dem Schließen eines Fensters aufgerufen. Sinnvolle Aktionen an dieser Stelle können sein: Zeichnen einer sich zusammenziehenden Box oder Freigeben eines Zugriffspfads etc. Beispiele sind in allen Modulen angegeben, welche Fenster benutzen.

VOID (*delete) (WINDOWP window);

Die Funktion wird unmittelbar vor dem Löschen eines Fensters aufgerufen. Sinnvolle Aktionen an dieser Stelle können sein: Abspeichern des aktuellen Fensterinhalts bzw. Dokumentes.

VOID (*draw) (WINDOWP window);

Die Funktion wird aufgerufen wenn der Fensterinhalt gezeichnet werden soll. Dies ist z.B. nach einer Redraw-Nachricht von GEM der Fall. Diese Funktion ist einer der Hauptgründe für das Einführen der indirekten Funktionsaufrufe. Der Window-Manager kann nicht wissen, welcher Inhalt sich in den Fenstern befindet. Er berücksichtigt allerdings eine eventuell vorhandene Menüzeile und die Rechteckliste. Dann springt er für jedes freiliegende Rechteck die Funktion „draw" an. Das aktuelle Clipping wurde vorher eingestellt und kann durch Inspizieren der globalen Variablen „clip" (siehe GLOBAL.H) in den Zeichenalgorithmus eingehen. Beispiele sind in allen Modulen zu finden, die Fenster benutzen.

VOID (*arrow) (WINDOWP window, WORD dir, LONG oldpos, LONG newpos);

Die Funktion wird aufgerufen, wenn der Benutzer entweder einen Pfeil in einem Fenster angeklickt oder den Schieber bewegt hat. „dir" gibt die Richtung an, also entweder HORIZONTAL oder VERTICAL (siehe WINDOWS.H). „oldpos" bezeichnet die alte Position, „newpos" die neue Position im Dokument. Abhängig von diesen Werten können nun verschiedene Dinge durchgeführt werden, z.B. das Vorrücken eines internen Cursors für Textzeilen oder Datensätze. Beispiele finden sich in allen Modulen, welche Fenster mit Schiebern benutzen.

VOID (*snap) (WINDOWP window, RECT *new, WORD mode);

Die Funktion wird aufgerufen, wenn das Fenster auf eine bestimmte Position oder Größe einrasten soll. Dies ist immer nach einem Bewegen oder Vergrößern/Verkleinern des Fensters der Fall. Dadurch kann die Applikation bestimmen, ob bestimmte Pixelpositionen eingehalten werden sollen.

Es hat sich gezeigt, daß z.B. die Textausgabe wesentlich schneller vonstatten geht, wenn sie an einer (horizontalen) Position beginnt, die durch 8 teilbar ist, also an den Positionen 0, 8, 16, usw. Außerdem erscheint es bisweilen sinnvoll, nur eine glatte Anzahl von z.B. Zeichen in einer Zeile darzustellen, damit man einen schnelleren Algorithmus beim Zeichnen anwenden kann. Schließlich muß die Menüzeile innerhalb eines Fensters immer an geraden Y-Postionen stehen, da sonst die grau eingestellten Menüs mit anderen Bitmustern grau gezeichnet werden, was bei einigen Buchstaben merkwürdig aussieht. Zu diesen Buchstaben gehören auch die Zeichen für Control (^) und Alternate (Raute).

VOID (*objop) (WINDOWP window, SET objs, WORD action);

Die Funktion wird aufgerufen, wenn innerhalb eines Fensters Objekte selektiert waren und eine der drei Standardfunktionen „Öffnen", „Info" und „Hilfe" (im Modul MENU) gewählt wird. „objs" ist die Menge von Objekten, die selektiert ist, „action" kann einen der drei folgenden Werte annehmen:

OBJ_OPEN: Die Objekte werden zum Öffnen angefordert.
OBJ_INFO: Es wird Information über die Objekte verlangt.
OBJ_HELP: Es wird eine Hilfemeldung über die Objekte verlangt.

Da es sich immer um eine Menge von Objekten handelt, können durchaus mehrere selektierte Objekte auf einmal geöffnet werden. Dies ist zum Beispiel auch beim Macintosh Finder der Fall, wo mehrere Dateien auf einmal selektiert werden können und beim Aufruf von Info die Information über alle Objekte erscheint. Beim GEM Desktop ist dies nicht der Fall. Beispielaufrufe finden sich im Modul MENU.

```
WORD (*drag) (WINDOWP src_window, WORD src_obj,
              WINDOWP dest_window, WORD dest_obj);
```

Wenn ein Objekt von einem Fenster in ein anderes geschoben wird, so wird diese Funktion vom Window-Manager aufgerufen. Sie wird nur benötigt, wenn Objekte in ein Fenster geschoben werden können und dies eine Wirkung hat, wie zum Beispiel bei geöffnetem Papierkorbfenster und einem Ziehen einer Datei in dasselbe. Die Funktion wird noch ausführlicher bei der Beschreibung der Funktion „drag_to_window" erklärt. „src_window" gibt das Ursprungsfenster an, aus dem ein Objekt gezogen wurde, „src_obj" gibt die Nummer des Objekts an, welches verschoben wurde. „dest_window" gibt das Zielfenster und „dest_obj" das Zielobjekt an, falls ein solches existiert. Als Rückgabewert kann ein WORD angegeben werden. Einige Werte (Konstanten) sind für Rückgabewerte bereits vordefiniert (siehe WINDOWS.H). Sie bedeuten:

DRAG_OK: Die Drag-Operation ist ok.
DRAG_SWIND: Das Objekt wurde innerhalb des gleichen Fensters verschoben.
DRAG_SCLASS: Das Objekt wurde auf ein Fenster der gleichen Klasse gezogen, also z.B. von einem Dateifenster in ein anderes, aber nicht von einem Dateifenster in ein Papierkorbfenster.
DRAG_NOWIND: Das Zielfenster, in welches das Objekt gezogen wurde, gehört nicht zum gleichen Prozeß, kann also z.B. ein Fenster von einem Deskaccessory sein. In einer Multitasking-Umgebung sollte dem fremden Prozeß mittels „wind_apfind" eine Nachricht gesendet werden.
DRAG_NORCVR: Das Zielfenster gehört zwar zum eigenen Prozeß, stellt aber keine Routine zur Verfügung, um Drag-Operationen auszuwerten.
DRAG_NOACTN: Das Zielfenster besitzt zwar ein Routine, welche auf eine Drag-Operation reagiert, kann aber mit dem Objekt nichts anfangen.

```
VOID (*click) (WINDOWP window, MKINFO *mk);
```

Bei jedem Klick in ein Fenster wird diese Funktion aufgerufen. Dabei werden genaueste Informationen über den Status der Maus und der Tastatur in der Variablen „mk" übergeben. Die Struktur MKINFO (Mouse-Keyboard-Info) enthält die Mauskoordinaten, den Status der Knöpfe und der Shift-Tasten. Das Fenster kann dann auf einen Einfach-, einen Doppelklick oder eine Ziehoperation reagieren. Dabei können selektierte Objekte hervorgehoben werden. Beispiele finden sich in den Modulen CLIPBRD, DESKTOP und POWER.

```
VOID (*unclick) (WINDOWP window);
```

Wurde ein Fenster angeklickt, so waren dort eventuell Objekte selektiert. Wird nun in ein anderes Fenster geklickt, so können diese Objekte vorher deselektiert werden. Das geschieht mit der Funktion „unclick". Beispiele finden sich in den Modulen CLIPBRD und DESKTOP.

BOOLEAN (*key) (WINDOWP window, MKINFO *mk);

Wenn der Benutzer eine Taste drückt, so ordnet der Window-Manager diese dem obersten Fenster zu. Dabei wird an dieses das gesamte Informationspaket in der Struktur „mk" übergeben, ähnlich wie es bei „click" der Fall ist. Zu den Informationen zählen der ASCII-Code, der Scan-Code, sowie der Status der Shift-Tasten.

Das oberste Fenster kann nun entscheiden, ob es die Taste für sich verbrauchen will oder nicht. Im ersten Fall wird als Rückgabewert TRUE angegeben. Für den Fall, daß die Routine FALSE zurückgibt, sucht der Window-Manager das nächstobere Fenster und bietet diesem die Taste an. Dieses kann nun ebenfalls entscheiden, ob es auf die Taste reagieren möchte. Das letzte Fenster, das die Taste dann bekommt, ist (bei regulären Applikationen) der Desktop. Beispiele werden in den Modulen CLIPBRD, DESKTOP, EDIT, GRAF und POWER angegeben.

VOID (*timer) (WINDOWP window);

Die Funktion wird angesprungen, wenn die Zeit verstrichen ist, die in der Variablen „milli" (s.o) angegeben ist. Dadurch kann das Fenster eine Aktion ausführen. Falls dies mit mehreren Fenstern gemacht wird, entsteht der Eindruck von Multitasking innerhalb eines Programms. Ein Beispiel ist im Modul GRAF angegeben.

VOID (*top) (WINDOWP window);

Wenn ein Fenster nach oben gebracht wird, so wird diese Funktion angesprungen. Zwei Gründe kann es dafür geben. Zunächst kann eine entsprechende Meldung vom Screen-Manager des GEM kommen. In diesem Fall wird die Nachricht WM_TOPPED an die Applikation geschickt. Der zweite Fall ergibt sich beim Schließen eines Fensters. Das vorher zweitoberste Fenster kommt nun nach oben. Dann wird vom Window-Manager die Funktion angesprungen. In dieser Funktion könnte z.B. das Restaurieren des Fensters aus einem zuvor geretteten Puffer geschehen.

VOID (*untop) (WINDOWP window);

Die Gegenfunktion von oben wird angesprungen, wenn ein Fenster nach unten kommt. Dafür kann es mehrere Gründe geben. Zunächst kann eine entsprechende Meldung vom Screen-Manager des GEM kommen. In diesem Fall wird die Nachricht WM_UNTOPPED an die Applikation geschickt. Die funktioniert aber erst ab GEM 2.X. Die Meldung kommt genau dann, wenn ein anderes Fenster mittels „WF_TOP" nach oben gebracht wird. Der im alten GEM fehlende Aufruf wird dabei vom Window-Manager nachgeholt. Der zweite Grund kann das Öffnen eines Fensters sein. In diesem Fall wird das oberste aktive Fenster inaktiv, und die Funktion wird angesprungen. In

dieser könnte z.B. das Retten des Fensterinhaltes realisiert sein, bevor andere Fenster sich darüber legen und diesen zerstören.

VOID (*edit) (WINDOWP window, WORD action);

Die Funktion wird angesprungen, wenn vom Edit-Menü eine der Funktionen Undo, Cut, Copy, Paste, Clear oder Select All aufgerufen wird. „action" kann dabei die Werte annehmen:

DO_UNDO : Die letzte Funktion soll rückgängig gemacht werden.
DO_CUT : Der markierte Block soll ausgeschnitten werden.
DO_COPY : Der markierte Block soll kopiert werden.
DO_PASTE : Der Puffer soll eingefügt werden.
DO_CLEAR : Der markierte Block soll gelöscht werden.
DO_SELALL : Das gesamte Dokument soll markiert werden.
DO_EXTERNAL: Die Operationen beziehen sich auf das GEM-Clipboard.

Was das Fenster nun im einzelnen durchführt, bleibt seine Sache. Ob die Operationen möglich sind, hängt davon ab, ob der Test über die Funktion „test" (s.o.) erfolgreich war.

BOOLEAN (*showinfo) (WINDOWP window, WORD icon);

Die Funktion wird aufgerufen, wenn der Benutzer das Menü „Info" anwählt. Dabei gibt es zwei Möglichkeiten. Entweder ist ein Objekt angewählt (invertiert), dann wird das selektierte Fenster über die Funktion „objop" angesprungen (s.o). Dieses Fenster entscheidet dann, welche Funktionen für die angewählten Objekte angesprungen werden müssen. Ist kein Objekt in einem Fenster selektiert, so wird die Info-Funktion des obersten Fensters angesprungen. „window" zeigt in diesem Fall auf das oberste Fenster. „icon" gibt die Objektnummer des angewählten Objekts an, z.B. ein Piktogramm. Beispiele zu dieser Funktion finden sich in den Modulen CLIPBRD, DESKTOP und MENU.

BOOLEAN (*showhelp) (WINDOWP window, WORD icon);

Die Funktion hat dieselbe Bedeutung wie „showinfo", jedoch wird sie angesprungen, wenn der Benutzer das Menü „Hilfe" anwählt. Ansonsten gilt das schon oben gesagte.

Ein Modul muß nun die oben erklärte Struktur WINDOW mit konkreten Werten füllen, damit der Window-Manager seine Arbeit verrichten kann. Besonders wenn mehrere Fenster geöffnet werden können, müssen diese zuerst kreiert werden. Jedes Modul, das Fenster öffnen kann, sollte deshalb die schon weiter oben beschriebenen Grundfunktionen besitzen. Sie sollen an dieser Stelle noch einmal mit den Parametern erklärt werden. Statt des Wortes „module" sollte man sich ein konkretes Objekt vorstellen, z.B. „image" für die Fenster, welche Image-Dateien anzeigen können.

– Kreieren eines Fensters

GLOBAL WINDOWP crt_module (OBJECT *obj, OBJECT *menu, WORD icon, ...);

„obj" gibt einen Objektbaum an, der sich im Fenster befinden kann (siehe 6.3).

„menu" gibt die Menüzeile an, die sich im Fenster befinden kann (siehe 6.3).

„icon" gibt das Piktogramm an, aus welchem das Fenster entstehen wird, falls es existiert.

Die Funktion liefert einen Zeiger auf eine Fensterstruktur zurück (siehe 6.3). Alle Variablen der Fensterstruktur WINDOW sollten in dieser Funktion initialisiert werden. Falls die drei Parameter von oben nicht ausreichen, können auch weitere Parameter definiert werden. Im Modul EDIT finden wir z.B. noch die Parameter „filename" und „size". Sie geben den Dateinamen und die maximale Größe an, die für die Datei reserviert werden soll, welche geöffnet wird. Beispiele dafür finden sich in allen Modulen, die Fenster öffnen können.

– Öffnen eines Fensters (oder einer Dialogbox etc.)

GLOBAL BOOLEAN open_module (WORD icon, ...);

„icon" hat dieselbe Bedeutung wie oben.

Die Funktion liefert TRUE, wenn das Fenster geöffnet werden konnte, FALSE sonst. Beim Öffnen eines Fensters können auch noch weitere Parameter übergeben werden. Wichtig kann z.B. ein Dateiname sein, der geladen werden soll. Im Modul EDIT wird dies beispielhaft gezeigt.

– Information eines Fensters

GLOBAL BOOLEAN info_module (WINDOWP window, WORD icon);

„window" gibt das Fenster an, von welchem Information angezeigt werden soll.

„icon" gibt das Piktogramm an, von welchem Information gezeigt werden soll.

Die Funktion sollte folgendermaßen ablaufen (siehe auch „info_edit" im Modul EDIT). Ist ein Piktogramm über den Parameter „icon" angegeben („icon" != NIL), so sollte das entsprechende Fenster dazu gesucht werden. Ist dies nicht der Fall, so sollte der Parameter „window" benutzt werden, um die zum zugehörigen Fenster gehörende Information anzuzeigen. Durch diesen Algorithmus ergibt sich der Effekt, daß durch das Modul

MENU zunächst ein aktives (invertiertes) Objekt berücksichtigt werden kann, dann erst das oberste Fenster.

– Hilfe eines Fensters

GLOBAL BOOLEAN help_module (WINDOWP window, WORD icon);

„window" gibt das Fenster an, zu dem Hilfe angezeigt werden soll.

„icon" gibt das Piktogramm an, zu dem Hilfe gezeigt werden soll.

Die Funktion sollte folgendermaßen ablaufen. Ist ein Piktogramm über den Parameter „icon" angegeben („icon" != NIL), so sollte das entsprechende Fenster dazu gesucht werden. Ist dies nicht der Fall, so sollte der Parameter „window" benutzt werden, um die zum zugehörigen Fenster gehörende Hilfe anzuzeigen. Durch diesen Algorithmus ergibt sich der Effekt, daß durch das Modul MENU zunächst ein aktives (invertiertes) Objekt berücksichtigt werden kann, dann erst das oberste Fenster.

Wie arbeitet nun der Window-Manager? Allgemein gilt folgendes: Eine Applikation kreiert verschiedene Fenster über ein oder mehrere Module („crt_module"). Die Fenster werden beim Window-Manager in eine Liste eingetragen. Er weiß dann immer, welche Fenster sich in welcher Reihenfolge offen oder geschlossen auf dem Bildschirm befinden. Zu jedem Fenster gibt es eine Reihe von Funktionen, die oben beschrieben wurden.

Trifft nun eine Nachricht ein, so daß z.B. ein Fenster geöffnet werden muß, so wird einfach die Funktion „open_module" des entsprechenden Moduls aufgerufen. Dieses ruft dann seinerseits eine Funktion des Window-Managers auf, nämlich „open_window" (s.u.). Diese erledigt wiederum alle Aktionen, welche GEM benötigt, um das Fenster auf den Bildschirm zu bringen.

Durch die bekannte Struktur WINDOW kann die Funktion „open_window" aber noch mehr tun. Handelt es sich beispielsweise um ein Fenster mit „kind" 0 (= DESK), so wird statt des Öffnens des Fensters der Desktop angemeldet. Vor dem eigentlichen Öffnen wird eventuell ein Fenster, welches verdeckt wird, dazu veranlaßt, eine Aktion durchzuführen („untop"). Dann wird für das zu öffnende Fenster die Routine „open" aufgerufen, um eine Aktion beim Öffnen auszuführen (z.B. eine Box zu zeichnen).

Erst dann wird das Fenster tatsächlich geöffnet. Dabei werden alle Attribute des Fensters berücksichtigt. Name und Infozeile werden gesetzt. Falls Schieber existieren, werden diese ebenfalls gesetzt.

Bei anderen Fensteraktionen wird ebenfalls eine Reihe von Aktionen durchgeführt. Das Schema ist immer das gleiche. Der Window-Manager versucht, möglichst viele Aktionen, die allgemein für alle Fenster gelten, auszuführen. Der Unterschied ergibt sich jeweils durch die Angabe der Funktionen in der Struktur WINDOW, die vom Window-Manager bei den entsprechenden Aktionen (indirekt) aufgerufen werden. Ein weiterer

Unterschied ergibt sich durch die Inhalte der Variablen in der Struktur WINDOW. Durch richtige Besetzung der Variablen „doc" kann die Schiebergröße automatisch berechnet werden. Bei Angabe der Variablen „object" bzw. „menu" wird die Objektbaumverwaltung bzw. die Menüverwaltung des Window-Managers dazu veranlaßt, vollautomatisch die entsprechenden Aktionen auszuführen.

Das Modul WINDOWS mit seiner Schnittstelle wird weiter unten bei der Modulbeschreibung erläutert. Dort wird für jede der 52 Funktionen erkärt, wie und wann sie aufgerufen werden sollte.

## 6.4 Programmierumgebungen für SCRAP

Bevor auf die Modulbeschreibung eingegangen wird, soll die Programmierumgebung für die Beispielapplikation erklärt werden. In Kapitel 3 wurde bereits erklärt, wie die einzelnen Dateien im Verzeichnis MISC auf die Inhaltsverzeichnisse der Compiler übertragen werden sollen. Die dort erklärten Schritte sollten auf jeden Fall zuerst durchgeführt werden. Nun geht es darum, für jedes Betriebssystem die Umgebung herzustellen, mit der am besten gearbeitet werden kann.

Für alle Betriebssysteme gilt zunächst einmal: Kopieren des Ordners SCRAP der Begleitdiskette auf irgendeine Partition in irgendeinen Ordner der Festplatte. Falls Sie keine Festplatte besitzen, sollten Sie den Ordner auf eine Arbeitsdiskette kopieren. Falls Sie keine zwei Laufwerke haben, können Sie den Ordner auch auf eine Ramdisk überspielen. Falls Sie nur ein Laufwerk und nicht genügend Speicher haben, sollten Sie erwägen, ob Sie sich nicht einen Massenspeicher wie eine Festplatte zulegen sollten. Das erspart Ihnen beim ernsthaften Arbeiten einige graue Haare bzw. wertvolle Tage Ihres Lebens.

Als Beispiel haben wir nun den Ordner SCRAP auf die Partition D: der Festplatte kopiert, so daß dort nun der Ordner

D:\SCRAP

mit allen Dateien vorliegen soll. Falls Ihr Ordner sich auf Laufwerk B: befindet, müssen Sie sich an allen Stellen, an welchen D:\SCRAP steht, ein B:\SCRAP vorstellen.

Da die Resource-Dateien nicht direkt kompatibel sind, kopieren Atari-Besitzer die Dateien des Ordners A:\SCRAP\M68000.RSC ebenfalls in den Ordner D:\SCRAP. Es sind dies die Dateien

SCRAP.DEF
SCRAP.RSC

Besitzer eines Rechners mit Intel-Prozessor (Betriebssysteme MS-DOS oder FlexOS) kopieren die Dateien des Ordners A:\SCRAP\I8086.RSC ebenfalls in den Ordner D:\SCRAP. Es sind dies die Dateien

SCRAP.DFN
SCRAP.RSC

Nun müssen wir uns entscheiden, mit welchem Compiler wir arbeiten wollen. Für jedes Betriebssystem werden die einzelnen Compiler aufgeführt. Auf der Begleitdiskette befindet sich zu jedem Betriebssystem ein Ordner mit dem entsprechenden Namen, d.h. GEMDOS, MS-DOS und FLEXOS. In diesen Ordnern befinden sich wiederum Ordner für jeden Compiler des entsprechenden Betriebssystems. Von diesem müssen jeweils Dateien auf die Arbeitsdiskette/Festplatte in den Ordner SCRAP kopiert werden. Wir gehen im folgenden davon aus, daß sich die Begleitdiskette in Laufwerk A befindet.

### 6.4.1 GEMDOS

Wenn Sie auf einem Atari ST arbeiten, können Sie unter mehreren Compilern wählen. Die Auflistung erfolgt hier alphabetisch, nicht nach Beliebtheit.

a) Digital Research C

Wenn Sie noch mit diesem ersten C-Compiler für den Atari ST arbeiten, so sollten Sie zusätzlich einen Kommandointerpretierer haben (COMMAND.PRG), damit Sie wenigstens Batch-Dateien ausführen können. Das gleiche gilt für den Nachfolger ALCYON C. Sie kopieren nun

```
A:\GEMDOS\DR_C\C.BAT        -> D:\SCRAP
A:\GEMDOS\DR_C\L.BAT        -> D:\SCRAP
A:\GEMDOS\DR_C\C_ALL.BAT -> D:\SCRAP
```

Um nun ein Modul zu kompilieren, geben Sie (z.B. für GLOBAL)

```
c global
```

ein. Die Batch-Datei „C.BAT" startet dann sämtliche Phasen des Compilers und des Assemblers. Am Ende wird noch der Archiver aufgerufen, da an den Linker später nicht so viel Module übergeben werden können. Aus diesem Grund werden alle Module in einer Bibliothek gehalten, die dann zusammengelinkt werden.

Wollen Sie danach linken, geben Sie einfach

```
l
```

ein. Dann wird der Linkvorgang gestartet.

Müssen Sie alle Module übersetzen, dann können Sie

```
c_all
```

eingeben. Dann werden alle Module übersetzt und anschließend gelinkt. Vorausgesetzt Sie haben genügend Platz auf Ihrer Festplatte oder Ramdisk (der Assembler öffnet ungefähr zehn Zwischendateien), so können Sie allerdings getrost ins Kino oder ins Theater gehen. Der DR C-Compiler bzw. der Alcyon C-Compiler gehören mit Ihren vier Durchläufen und langen Übersetzungszeiten zu den Schnecken unter den Compilern. Am Ende liegt das ausführbare Programm SCRAP.PRG vor, welches Sie wie gewohnt vom Desktop starten können.

b) Laser C

Im Gegensatz zum DR C-Compiler gehört der Laser C (ab Version 1.2) zu den schnellsten Compilern auf dem Atari ST. Glücklicherweise hat er außerdem ein integriertes Make, welches Unix-kompatibel ist. Der Quelltext-Debugger macht außerdem das Suchen von Fehlern zum Kinderspiel. Das Einrichten der Entwicklungsumgebung beschränkt sich hier auf das Überspielen einer Datei. Sie kopieren

```
A:\GEMDOS\LASER_C\MAKEFILE -> D:\SCRAP
```

Wenn Sie nun den Laser C starten, müssen Sie zunächst die Make-Datei laden, indem Sie im Menü „Make" den Menüpunkt „Set Make..." anwählen. Dann erscheint eine Dateiauswahl-Box. Dort suchen Sie nach D:\SCRAP\MAKEFILE und wählen es aus. Nachdem die Datei geladen wurde, wählen Sie im Menü „Make" den Menüpunkt „M scrap.prg" an. Nun wird die komplette Applikation „gemacht". Zum Starten wählen Sie unter dem Menü „Execute" den Menüpunkt „Other..." aus. Bei der folgenden Dateiauswahl-Box suchen Sie SCRAP.PRG und klicken es doppelt an.

Falls Sie ein oder mehrere Module ändern, reicht es, immer wieder „M scrap.prg" anzuwählen. Make garantiert, daß alle Dateien, die betroffen sind, immer wieder neu compiliert werden. Voraussetzung ist, daß das Datum und die Uhrzeit in Ihrem Rechner richtig gesetzt ist. Die Abhängigkeiten, welche in der Makedatei für jedes Modul definiert sind, ergeben sich genau aus den „include"-Anweisungen in den C-Dateien. Die Datei EVENT.C beginnt beispielsweise mit den Zeilen

```
#include ''import.h''
#include ''global.h''
#include ''windows.h''
#include ''desktop.h''
#include ''menu.h''

#include ''export.h''
#include ''event.h''
```

Die Abhängigkeit von „event.o“ in der Makedatei definiert sich zu

event.o: $(H) windows.h desktop.h menu.h event.h

wobei $(H) ein Makro ist, welches vorher zu

H = import.h export.h global.h

definiert wurde. Eingesetzt ergibt sich

event.o:import.h export.h global.h windows.h desktop.h menu.h event.h

Die Dateien im Makro $(H) sind maßgebend für jede Quelldatei, da jede Quelldatei von ihnen abhängt. Die anderen Dateien sind exakt die Dateien, welche in den „include“-Anweisungen auftreten. Natürlich hängt „event.o“ auch von der Datei „event.c“ ab, doch diese Abhängigkeit ist implizit und muß nicht speziell definiert werden.

In der Makedatei können außerdem noch Compileroptionen (CFLAGS) und Linkeroptionen (LFLAGS) definiert werden, die jedoch hier ohne Bedeutung sind (siehe jedoch Mark Williams C).

c) Lattice C

Für das Arbeiten mit dem Lattice C-Compiler (ab Version 3.04) kopieren Sie

A:\GEMDOS\LATTICE\SCRAP.LNK -> D:\SCRAP

Am besten stellen Sie auch als Linker den schnellen FLINK ein, indem Sie in der Datei MENU.INF die Zeile

LINK = FLINK {path}\{file} {linker_opts} ? WAIT : WAIT

aufnehmen, d.h. LINK durch FLINK austauschen.

Vor dem Compilieren müssen Sie die Compiler- und Linkeroptionen ändern. Starten Sie die Shell MENU+.PRG und gehen Sie im Menü „Options“ auf den Menüpunkt „PHASE1_OPTS“. Tragen Sie

-n -I{path}\ -I..\HEADERS\

ein. Dies garantiert, daß alle Include-Dateien gefunden werden. Ebenso ändern Sie die Linker-Optionen unter dem Menüpunkt „LINKER_OPTS“ und tragen

-with SCRAP.LNK -debug -nolist

ein. Zum Compilieren eines Moduls wählen Sie dieses zunächst aus. Das geschieht über das Menü „File" und den Menüpunkt „Choose C". Daraufhin wählen Sie mittels der Dateiauswahl-Box die C-Quelldatei aus. Dann starten Sie den Compiler über den Menüpunkt „COMPILE" im Menü „Tools". Nachdem Sie auf diese Weise alle Module übersetzt haben, müssen Sie noch linken. Dazu wählen Sie die Datei SCRAP.H über den Menüpunkt „Choose H". Der Grund dafür liegt darin, daß die ausführbare Programmdatei den gleichen Rumpf bekommt, wie die ausgewählte Datei, also SCRAP. Dann wählen Sie „LINK" im Menü „Tools". Nach dem Linken können Sie das Beispielprogramm starten, indem Sie den Menüpunkt „RUN PROGRAM" im Menü „Tools" aufrufen.

d) Mark Williams C

Dem Mark Williams C-Compiler (ab Version 3.0.9) merkt man schnell an, daß er aus dem Unix-ähnlichen Betriebssystem Coherent stammt. Für den Entwickler bedeutet dies, daß er keine Probleme hat, damit zu arbeiten, wenn er sich in Unix auskennt. Der Vorteil der Entwicklungsumgebung liegt eindeutig im Bereich des Unix-kompatiblen Make-Mechanismus. Wie beim Laser C (s.o.) beschränkt sich das Einrichten der Entwicklungsumgebung auf das Überspielen von zwei Dateien. Sie kopieren

```
A:\GEMDOS\MW_C\MAKEFILE -> D:\SCRAP
A:\GEMDOS\MW_C\STRING.H -> C:\MWC\INCLUDE
```

Die Datei STRING.H ist eine von uns entwickelte Headerdatei, in der die „Prototypen" der String- und Memory-Funktionen aufgeführt sind. Diese Datei wird in IMPORT.H verlangt, und der Mark Williams Compiler gehört zu den wenigen Compilern, die diese Headerdatei noch nicht in ihr Repertoire aufgenommen haben. In unserem Beispiel gehen wir davon aus, daß der Mark Williams Compiler auf Laufwerk C in dem Ordner MWC eingerichtet wurde. Dort befindet sich auch der Ordner INCLUDE für die Headerdateien.

Die Makedatei ist vollständig kompatibel zum Laser C, so daß Sie das dort Gesagte nachlesen sollten. Lediglich die beiden Flags CFLAGS zum Compilieren und LFLAGS zum Linken sind hier verschieden. Sie lauten

```
CFLAGS = -VPEEP
LFLAGS = -x -s -laes -lvdi
```

Für den Compiler wird also der Peephole-Optimizer eingeschaltet. Diese Option hat erst ab Version 3.x ihre Gültigkeit. Besitzen Sie eine ältere Version, so müssen Sie CFLAGS auf leer setzen. Beim Linken werden die globalen Symbole sowie die Symboltabelle entfernt (-x -s) und die Bibliotheken von AES und VDI hinzugelinkt. Man hätte das auch durch die Compileroption -VGEM erreichen können, jedoch wird dann ein anderes Startup-Modul benutzt. Der Linker soll aber das von uns modifizierte CRTS0.O benutzen, da dieses wiederum fähig ist, Applikationen und Accessories gleichzeitig zu handhaben.

Das Übersetzen von SCRAP geschieht nun nach dem Starten von MSH.PRG dadurch, daß zunächst in das richtige Inhaltsverzeichnis gewechselt werden muß. Sie geben

```
cd d:\scrap
```

ein. Danach genügen die vier Buchstaben „make" und die Applikation wird hergestellt. Nach dem Linken können Sie das Programm durch den Aufruf

```
gem scrap
```

starten.

e) Turbo C

Der Turbo C-Compiler (ab Version 1.0) ist der einzige ANSI-C-Compiler auf dem Atari ST. Er gehört zu den schnellsten Compilern und erzeugt den absolut besten Code. Seine Entwicklungsumgebung ist als herausragend zu bezeichnen. Allerdings fehlt noch ein guter Debugger. Auch hat er kein Make, aber dafür besitzt er etwas Ähnliches. Hier gibt es die sogenannten Projektdateien. Es genügt das Überspielen einer Datei, nämlich

```
A:\GEMDOS\TURBO_C\SCRAP.PRJ -> D:\SCRAP
```

Starten Sie TC.PRG und wählen Sie im Menü „Projekt" den Menüpunkt „Select..." aus. Damit können Sie über eine Dateiauswahl-Box die Projektdatei SCRAP.PRJ auswählen. Zum Compilieren und Linken wählen Sie nur noch „Make SCRAP.PRJ" aus dem Menü „Projekt" aus. Aus dem gleichen Menü können Sie auch sofort „Run SCRAP.PRJ" auswählen. Dann wird nach dem Compilieren und Linken das Programm sofort gestartet.

Für alle Compiler unter Ataris GEMDOS gilt noch eine weitere Regel. Alle Make- und Projektdateien erzeugen am Ende das Programm SCRAP.PRG. Soll das Programm als Accessory gestartet werden, so ist es lediglich von SCRAP.PRG auf SCRAP.ACC umzubenennen. Danach muß es auf die Bootdiskette oder die Boot-Partition kopiert werden. Es erkennt beim Booten automatisch, daß es nun ein Accessory ist und reagiert entsprechend. Mit allen oben genannten Compilern außer dem Lattice C ist es außerdem noch möglich, das Programm, auch wenn es bereits auf SCRAP.ACC umbenannt wurde, vom Desktop aus zu starten. Dazu muß allerdings die Zeile

```
#G 03 FF    *.ACC@ @
```

in die Datei DESKTOP.INF aufgenommen werden. Dann werden alle Accessories nämlich ebenfalls durch Programm-Piktogramme angezeigt. Sie können dann ebenfalls gestartet werden. Durch entsprechende Abfragen in den Startup-Dateien der Compiler ist es möglich, ein solches Accessory auch als Programm richtig ablaufen zu lassen.

Für MS-DOS ist die Entwicklung von Accessories wegen des begrenzten Speicherraums von 640K nicht sinnvoll. Für OS/2 und FlexOS bei Verwendung von X/GEM sind die Zeiten der Accessories wegen der Multitasking-Umgebung glücklicherweise vorbei – man benötigt sie nicht mehr.

### 6.4.2 MS-DOS

Für das populäre Betriebssystem von Microsoft haben wir zwei Compiler ausgewählt, mit dem die Beispielapplikation ablauffähig ist. Auf jeden Fall benötigen Sie aber das Programmer's Toolkit von Digital Research, manchmal auch einfach PTK genannt. Zu diesem Toolkit werden neben dem Resource-Construction-Set und anderen Utilities auch die Quellen der Bindings für die diversen Bibliotheken mitgeliefert.

Glücklickerweise ist das Installieren des PTK von nun an ein Kinderspiel. Während man früher die Bindings noch selbst an den jeweiligen Compiler anpassen mußte, wurden sie nun so umgeschrieben, daß sie mit verschiedenen Compilern arbeiten. Dies haben wir der hervorragenden Arbeit von Robert Schneider zu verdanken, der in der deutschen Tochter von Digital Research GmbH in München für den technischen Support zuständig ist. Mit Hilfe eines GEM-Programms klickt man sich nur noch durch die Installierung und muß am Ende lediglich noch ein MAKELIB eingeben, dann werden die kompletten Bibliotheken aus den Bindings erstellt.

a) Microsoft C

Der Microsoft C-Compiler (inzwischen in der Version 5.1) ist sicher einer der Compiler mit der besten Codeerzeugung. Er ist allerdings nicht besonders schnell. Auch er besitzt ein Make, das sich von den Make-Programmen unter Unix etwas unterscheidet. Drei Dateien sollten Sie überspielen, und zwar

```
A:\MSDOS\MS_C\MK.BAT      -> D:\SCRAP
A:\MSDOS\MS_C\MAKEFILE    -> D:\SCRAP
A:\MSDOS\MS_C\SCRAP.LNK -> D:\SCRAP
```

Die Datei MK.BAT besteht eigentlich nur aus einer Befehlszeile. Sie lautet

```
make makefile
```

Beim Microsoft Make muß immer die Makedatei angegeben werden. Ein einfaches

```
make
```

wie unter Unix reicht hier nicht. Es ist viel einfacher „mk" einzugeben, als „make makefile" (2 Buchstaben gegenüber dreizehn).

Die Makedatei entspricht ungefähr den Unix-Konventionen. Allerdings muß man hier mehr Leerzeilen einfügen. Außerdem gibt es keine (externe) implizite Regeln. Sie muß man in der Make-Datei einfügen. So wird dort die Regel

```
.c.obj:
    cl -c $(CFLAGS) $*.c
```

aufgeführt, die angibt, wie eine C-Datei in eine OBJ-Datei überführt werden muß. Die Compiler- und Linker-Flags lauten hier

```
CFLAGS = -AL -Oas -Gs

LFLAGS = /ST:12288 /SE:512 /NOI /E

APP    = .app
```

Der Compiler wird dadurch dazu gebracht, im Large-Modell zu übersetzen und die Codegröße zu optimieren (alias-checking off, size optimization, stack checking off)

Bei den Linkflags werden die Stackgröße (hier 12 KB = 12288 Bytes) und die Anzahl der Segmente (hier 512) angegeben. Außerdem soll der Linker Groß- und Kleinschrift unterscheiden (NOI) und die entstehende Datei packen (E).

Das fertige Programm trägt das Suffix „.app", wird also „scrap.app" heißen. Dies ist die übliche Konvention für GEM-Applikationen.

Falls es passiert, daß der Speicher für eine große Applikation nicht mehr ausreicht, muß ein anderer Linker herangezogen werden, der auch verschachtelte Overlays zuläßt. Ein solcher Linker ist der PLINK86. Er erwartet natürlich eine andere Link-Control-Datei.

In der Makedatei müßte man dann die Zeile

```
link $(LFLAGS) @$(NAME).lnk
```

durch

```
plink86 @$(NAME).pl
```

ersetzen, falls die Link-Control-Datei das Suffix „pl" aufweist.

Das Compilieren und Linken geschieht nun einfach durch Eingabe von „mk". Damit werden das Make gestartet und die Datei SCRAP.APP hergestellt. Ein Starten des Programms ist vom Desktop aus möglich, wobei die Datei doppelt angeklickt wird. Eine andere Möglichkeit ist die Eingabe von

```
gem scrap
```

Dann wird die Applikation „scrap" statt des Desktops gestartet. Dazu muß sich SCRAP.APP aber im Ordner GEMAPPS befinden, und zwar auf der Partition der Festplatte, auf der sich das GEM-System befindet.

b) Turbo C

Für den Turbo C-Compiler (ab Version 1.5) haben wir zwei Möglichkeiten eingeplant. Zum einen kann man in der integrierten Entwicklungsumgebung arbeiten. Dort hat man einen rasant schnellen Turn-around-Zyklus. Compilieren, Fehler verbessern und linken geschieht mit atemberaubender Schnelligkeit. Es gibt aber auch einen Stand-alone Compiler und -Linker, die zusammen mit einem Make eine ebenfalls starke Entwicklungsumgebung bieten. In folgenden Beispielen wollen wir davon ausgehen, daß sich der Turbo C-Compiler auf der Festplatte C im Ordner \TC befindet. Für die integrierte Umgebung überspielen Sie

```
A:\MSDOS\TURBO_C\SCRAP.PRJ      -> D:\SCRAP
A:\MSDOS\TURBO_C\TCCONFIG.TC -> D:\SCRAP
```

Die Datei SCRAP.PRJ gibt die Projektdatei an. Sie enthält alle Dateinamen von SCRAP und den Namen „LTCGEM.LIB" (GEM-Bibliothek aus dem PTK). Die Datei TCCONFIG.TC enthält Einstellungen der integrierten Umgebung. So wird beispielsweise das Stack-Checking abgeschaltet, da sonst die benutzerdefinierten Objekte nicht richtig arbeiten. Außerdem werden bestimmte Warnmeldungen unterdrückt, die hier keine Bedeutung haben.

Das Übersetzen und Linken geschieht nun folgendermaßen: Starten der Turbo C Umgebung durch Eingabe von

```
tc
```

Dann muß die Projektdatei geladen werden. Dies geschieht über den Menüpunkt „Projekt name" im Menü „Projekt". Anschließend kann die Funktionstaste F9 gedrückt werden. Nun wird die Datei SCRAP.APP hergestellt. Falls ein Fehler auftritt, kann sofort auf die entsprechende Stelle gesprungen werden. Wird eine Datei geändert, reicht das Drücken von F9 zum erneuten Compilieren und Linken. Das Starten ist aus der integrierten Umgebung nicht möglich, da zuerst GEM aufgerufen werden muß. Nach Verlassen der Umgebung über Alt-X kann der GEM-Desktop aufgerufen und das Programm von dort gestartet werden. Wie auch schon oben erwähnt, ist aber auch ein Aufruf von

```
gem scrap
```

möglich. Auch hier muß sich SCRAP.APP im Ordner GEMAPPS befinden.

Die zweite Möglichkeit der Programmerstellung ist über einen Make-Mechanismus gegeben. Dazu überspielen Sie

```
A:\MSDOS\TURBO_C\MAKEFILE       -> D:\SCRAP
A:\MSDOS\TURBO_C\SCRAP.LNK      -> D:\SCRAP
A:\MSDOS\TURBO_C\BUILTINS.MAK   -> C:\TC
```

Die Makedatei entspricht der Unix-Konvention, so daß Sie oben unter GEMDOS die Erklärungen zum Laser C lesen sollten. Die Compiler- und Linkerflags lauten hier

CFLAGS = -ml -w-par

LFLAGS = /c /x

Der Compiler übersetzt alle Dateien im Large-Modell. Dabei wird die Warnung für nichtbenutzte Parameter abgeschaltet.

Der Linker wird dazu gebracht, Groß- und Kleinbuchstaben zu unterscheiden (c = case significant) und keine Mapdatei zu erzeugen (x).

Beim Linken lautet die Regel

```
$(NAME)$(APP): $(OBJS)
    tlink $(LFLAGS) @$(NAME).lnk
```

Es wird also der Turbo Linker „tlink" aufgerufen. Er holt sich die Namen der zu linkenden Dateien aus „scrap.lnk" (NAME = scrap).

Die Datei BUILTINS.MAK, die Sie oben ebenfalls überspielt haben, gibt die implizite Regel an, die benutzt werden soll, um eine C-Datei in eine Objekt-Datei überzuführen. Sie muß sich auf demselben Inhaltsverzeichnis wie das Programm MAKE.EXE befinden (hier C:\TC). Der Inhalt lautet

```
.c.obj:
    tcc -c $(CFLAGS) $*

.asm.obj:
    tasm /mx $*
```

Um also eine Objektdatei aus einer C-Quelle zu erstellen, wird der Turbo C-Compiler „tcc" aufgerufen. Er übersetzt nur (-c), übernimmt aber die CFLAGS aus der Makedatei. Das $* steht für die aktuelle, zu übersetzende Datei. Wird kein Suffix angegeben, so wird eine C-Datei angenommen. Die Regel für Assemblerprogramme (hier mit dem Turbo Assembler) wird der Vollständigkeit halber auch aufgenommen. Damit läßt sich das BUILTINS.MAK auch für das PTK verwenden, wo ja eine Assemblerdatei (FARDRAW.ASM) für benutzerdefinierte Objekte benutzt werden muß.

Durch die Eingabe von

make

wird nun der Mechanismus gestartet. Nach dem Linken liegt die komplette Applikation vor. Das Make garantiert, daß nur die Dateien neu übersetzt werden, die auch wirklich

geändert wurden. Das Starten der Beispielapplikation geschieht dann wie oben, mit dem Unterschied, daß man die integrierte Umgebung, in der man sich ja nicht befindet, auch nicht verlassen muß.

### 6.4.3 FlexOS

FlexOS ist ein Multiuser-, Multitasking-, Echtzeit-Betriebssystem von Digital Research und das in diesem Augenblick einzige System, auf dem X/GEM komplett läuft. Die Version für OS/2 soll allerdings noch in 1989 folgen. Die Applikation GEMDRAW wurde bereits auf der CeBIT '89 gezeigt. Für FlexOS konnte nur ein Compiler getestet werden.

a) High C

Der High C-Compiler von Metaware ist ein besonders strenger ANSI-Compiler. Beim Übersetzen unserer Beispielapplikation ließ es sich leider nicht vermeiden, sehr viele Warnmeldungen zu erhalten. Der Grund dafür liegt darin zu suchen, daß wir die Syntax auch auf Nicht-ANSI-Compiler abgestimmt haben. In diesem Punkt ist der High C-Compiler zu streng. Beispielsweise erzeugt folgendes Programmstück (hier mit Zeilennummern) eine Warnung:

```
1 #include <portab.h>
2
3 WORD func (WORD i);
4
5 WORD func (i)
6 WORD i;
7 {
8 }
```

Zunächst wird der Prototyp der Funktion „func" in Zeile 3 angegeben. Dann folgt die Implementierung. In Zeile 5 und 6 werden die Parameter in der üblichen Form angegeben. Die Semantik ist aber die gleiche. Trotzdem behauptet der Compiler, die Semantik aus Zeile 3 wäre in den Zeilen 5 und 6 überschrieben worden. Der Compiler sieht nur folgendes als richtig an:

```
1 #include <portab.h>
2
3 WORD func (WORD i);
4
5 WORD func (WORD i)
6 {
7 }
```

Diese Warnmeldungen kann man zwar abschalten, jedoch wird dann keine Garantie mehr übernommen, daß die Definition des Prototyps und die Definition der Funktion semantisch gleich sind.

Ähnliches gilt für die Prototypen von statischen Funktionen innerhalb einer Funktion, die der Compiler als „extern" deklariert haben will. Tut man dies, so verträgt es sich jedoch auch nicht mit anderen Compilern. Mit den Warnungen muß man leben. Die einzige Alternative zum Beseitigen wäre das Umschreiben aller Funktionsdefinitionen. Aber wehe, man muß auf einen anderen, eventuell einen Nicht-ANSI-Compiler umsteigen. Auf diesem würde sich dann das Programm nicht mehr ohne Fehler übersetzen lassen.

Vor dem Compilieren überspielen Sie

```
A:\FLEXOS\HIGH_C\MAKEFILE    -> D:\SCRAP
A:\FLEXOS\HIGH_C\SCRAP.INP   -> D:\SCRAP
A:\FLEXOS\HIGH_C\SCRAP.PRF   -> D:\SCRAP
```

Das Make ist nicht ganz Unix-kompatibel. So gibt es z.B. keine impliziten Abhängigkeiten. Aus diesem Grund muß für jedes Modul ein Makro angegeben werden, welches den Compiler startet. Es lautet:

```
COMPILE = hc $*.c $(CFLAGS)
```

Damit wird der High C-Compiler gestartet. Die CFLAGS sowie die LFLAGS werden hier nicht benötigt, sie sind leer. Der Compiler muß jedoch vor der Benutzung eingerichtet werden. Dazu gibt es das Hilfsprogramm MWCONFIG. Damit stellt man den Compiler in das Big-Modell um. Die Codeerzeugung kann man auf 80286 stellen, da FlexOS sowieso nur ab diesem Prozessor (und neueren Modellen) lauffähig ist.

Beim Linken wird der „link86" von Digital Research benutzt. Damit ergibt sich die Abhängigkeit

```
$(NAME)$(APP): $(OBJS)
    link86 $(LFLAGS) $(NAME).inp[i]
```

Die Datei „scrap.inp" wird als Inputdatei benutzt. Sie wurde oben schon überspielt und enthält vor allem den Initialisierungscode und die X/GEM-Library-Namen. Sie sieht folgendermaßen aus:

```
scrap[stack[add[4000]]] =
xgembig.186[nose,li],
...
xgemsrtl.186[s,li],
dosrtl.186[s,li],
hcbe.186[s,li]
```

Die drei Punkte stehen für eigene Module. Der Stack wird um 4 KB erhöht, falls rekursive Algorithmen auftauchen. Ansonsten berechnet der Compiler die Stackgröße. Es folgt die X/GEM-Library für das Big-Modell, die als erste gelinkt werden muß. Nach den Modulen folgen die Schnittstelle für die X/GEM „shared runtime library", die von allen Prozessen geteilt wird, sowie die DOS „runtime library" und die High C Big Library.

Das Übersetzen und Linken geschieht einfach mit dem Aufruf

```
make
```

Die gelinkte Applikation bekommt das Suffix 286. Danach kann entweder der X/GEM-Desktop mittels

```
xgem
```

aufgerufen werden und von dort das Programm SCRAP.286 gestartet werden. Alternativ dazu kann auch

```
xgem scrap
```

aufgerufen werden. Dann wird ohne Laden des Desktops sofort SCRAP.286 gestartet. Da sowohl normale Programme als auch GEM-Programme das gleiche Suffix aufweisen, unterscheidet X/GEM die Art des Programms dadurch, daß es nach der Resource-Datei (hier SCRAP.RSC) sucht. Ist sie nicht vorhanden oder eingebunden, so muß X/GEM mitgeteilt werden, daß es sich um eine Grafik-Applikation handelt. Dies geschieht mit der Text-Datei SCRAP.PRF. Sie enthält dann die Zeile

```
GraphicsMode = True
```

Dadurch wird sichergestellt, daß es sich um eine Grafik-Applikation handelt.

Da FlexOS ein Multitasking-Betriebssystem ist, kann während des Compilierens natürlich auch gleichzeitig getestet werden, indem einfach weitere Konsolen geöffnet und weitere Prozesse gestartet werden – ein Traum für alle auf Singletasking-Systemen programmierende Entwickler.

## 6.5 Modulbeschreibung

Im folgenden werden die 18 Module der Beispielapplikation SCRAP beschrieben. Zu jedem Modul wird zunächst die Aufgabe, dann die Schnittstellenbeschreibung geliefert. Diese befindet sich in der Datei mit dem Suffix „H". Falls für das Verständnis notwendig, werden auch interne Details der Schnittstellenfunktionen beschrieben. Neben dem Durcharbeiten dieses Kapitels empfiehlt es sich auch, die entsprechenden Quelldateien in einen Editor zu laden und parallel dazu auf dem Monitor zu haben. Als Alternative dazu könnten natürlich die Quelldateien auch ausgedruckt werden.

Zunächst soll aber noch einmal auf die beiden Dateien „import.h" und „export.h" eingegangen werden. Die erste wird benutzt, um Bezeichner aus anderen Modulen zu importieren, damit diese dem importierenden Modul bekannt werden. Die zweite dient dazu, alle Bezeichner aus einem Modul zu exportieren. Für Variablen wird hierbei der entsprechende Speicherplatz angelegt (s.o.).

```
/*********************************************************************/
/*                                                                   */
/* IMPORT.H                                                          */
/* Datum: 24/05/89                                                   */
/*                                                                   */
/*********************************************************************/

#ifndef __IMPORT__
#define __IMPORT__

#include <stdio.h>
#include <string.h>
#include <portab.h>
#include <aes.h>
#include <vdi.h>

#if GEMDOS
#include <osbind.h>
#define Mavail() (LONG)Malloc (-1L)
#endif

#if MSDOS | OS2 | FLEXOS
#include <gemdos.h>
#include <dosbind.h>
#endif

#if ANSI
#include <stdlib.h>
#else
#define abs(x)      ((x) <  0  ? -(x) : (x))  /* Absolut-Wert */
#define labs(x)     abs (x)                   /* Langer Abs-Wert */
#endif

/****** DEFINES ******************************************************/

#ifdef GLOBAL
#undef GLOBAL
#endif

#define GLOBAL EXTERN

#if LASER_C
#define strchr  index
#define strrchr rindex
#endif
```

```
#if HIGH_C
#ifdef NULL
#undef NULL
#define NULL 0L
#endif
#endif

#ifndef max
#define max(a,b)    (((a) > (b)) ? (a) : (b)) /* Maximum-Funktion */
#define min(a,b)    (((a) < (b)) ? (a) : (b)) /* Minimum Funktion */
#endif

#define odd(i)      ((i) & 1)                 /* ungerade */

/*********************************************************************/

#ifdef PASCAL_DEF
#define and         &&                  /* Für Pascal-Programmierer */
#define or          ||
#define xor         ^^
#define not         !
#define div         /
#define mod         %

#define bitand      &
#define bitor       |
#define bitxor      ^
#define bitnot      ~

#define loop        for (;;)
#define exitloop(e) if (e) break
#define nextloop(e) if (e) continue

#define repeat      do {
#define until(e)    } while (! (e))

#define begin       {
#define end         }

#define then

#define boolean     BOOLEAN
#define integer     WORD
#define longint     LONG
#define real        FLOAT
#define longreal    DOUBLE
```

```
#define type        typedef
#endif /* PASCAL_DEF */

/*********************************************************************/

#ifdef MODULA_DEF
#define AND         &&                  /* Für Modula-Programmierer */
#define OR          ||
#define XOR         ^^
#define NOT         !
#define DIV         /
#define MOD         %

#define BITAND      &
#define BITOR       |
#define BITXOR      ^
#define BITNOT      ~

#define LOOP        for (;;)
#define EXITLOOP(e) if (e) break
#define NEXTLOOP(e) if (e) continue

#define REPEAT      do {
#define UNTIL(e)    } while (! (e))

#define BEGIN       {
#define END         }

#define WHILE(e)    while (e) {
#define IF(e)       if (e) {
#define THEN
#define ELSE        } else {
#define ELSIF(e)    } else if (e) {

#define CASE(e)     switch (e) {
#define OF

#define RETURN      return

#define INTEGER     WORD
#define LONGINT     LONG
#define CARDINAL    UWORD
#define LONGCARD    ULONG
#define REAL        FLOAT
#define LONGREAL    DOUBLE
#define BITSET      UWORD
#define LONGBITSET  ULONG
```

```
#define TYPE        typedef
#endif /* MODULA_DEF */

/*********************************************************************/

#endif /* _IMPORT_ */
```

Zunächst wird das Macro

__IMPORT__

definiert. Damit wird gewährleistet, daß die Datei nur maximal einmal vom Präprozessor durchlaufen wird. Wird sie ein zweitesmal eingelesen, so ist __IMPORT__ bereits definiert, so daß der Rest übersprungen wird. Diese Methode benutzen wir bei allen Headerdateien. Hat man nämlich verschachtelte include-Anweisungen in seinem Quelltext, dann ist nicht mehr unbedingt gewährleistet, daß jede Datei nur einmal komplett eingelesen wird. Wird eine Headerdatei jedoch öfters eingebunden, so hat der Compiler etwas dagegen, wenn beispielsweise eine Deklaration ein und desselben Typs mehr als einmal auftritt.

Danach werden zunächst fünf Standarddateien eingelesen. Die Datei „stdio.h" befindet sich auf dem Inhaltsverzeichnis zum jeweiligen Compiler, meist INCLUDE\. Dasselbe gilt für „string.h". Einzige Ausnahmen bilden der Mark Williams C-Compiler und der Laser C-Compiler. Beim MWC haben wir die Datei „string.h" selbst erstellt. Sie muß, wie bereits in 6.4.1 beschrieben, in das Verzeichnis INCLUDE\ des Mark Williams Compilers kopiert werden. Beim Laser C heißt die Datei fälschlicherweise „strings.h" und sollte in „string.h" umbenannt werden, um die Kompatibilität zu gewährleisten.

Nun folgt die Datei „portab.h", welche wesentlich zur Portabilität zwischen den Compiler-, GEM- und Betriebssystemversionen beiträgt. Viele werden sich nun fragen, warum diese nicht als erstes eingebunden wurde. Der Grund dafür liegt in der Definition von NULL, die in manchen „stdio"-Dateien definiert ist, in anderen wieder nicht. Die Definition aus „stdio.h" soll aber bevorzugt behandelt werden, da diese eventuell anders lautet. So ist NULL manchmal als

```
#define NULL 0L
```

oder als

```
#define NULL ((void *)0)
```

definiert. Würde zuerst „portab.h", dann „stdio.h" eingelesen, so würden manche Compiler diese Umdefinition als Warnung oder Fehler ansehen, da sie – zumindest literal gesehen – nicht gleich ist. Beim High C-Compiler von Metaware ist dies ein Problem, weswegen für ihn das NULL noch umdefiniert wird (s.u.).

Es folgen die Headerdateien „aes.h" und „vdi.h" mit all ihren Definitionen und Prototypen. Natürlich könnte man diese auch weglassen, falls man keine GEM-Applikation schreibt. Wir gehen aber davon aus, daß es sich um GEM-Applikationen handelt und lesen die beiden Dateien immer ein.

Nun folgt ein betriebssystemabhängiger Teil. Für GEMDOS (Atari ST) wird „osbind.h" eingelesen, in welchem Betriebssystemaufrufe definiert sind. Diese Datei existiert für jeden Compiler. Einzige Ausnahme bildet der Turbo C-Compiler, welcher die Datei leider „tos.h" genannt hat. Das liegt daran, daß sich die Entwickler nicht an bereits gültige Standards gehalten haben, sondern offensichtlich eigene Standards kreieren wollen. Um nicht ständig zwischen den Compilern unterscheiden zu müssen, sollten Sie das „tos.h" in „osbind.h" umbenennen. Zusätzlich wird das Makro „Mavail" definiert, welches den verfügbaren Speicher zurückgibt. Dies existiert bei MS-DOS und FlexOS (dos_avail), so daß wir es hier mit aufgenommen haben.

Für MS-DOS, OS/2 und FlexOS werden die Dateien „gemdos.h" und „dosbind.h" eingebunden. In „dosbind.h", welches zum Programmer's Toolkit gehört, finden sich verschiedene Betriebssystemaufrufe. Sie entsprechen zum größten Teil den entsprechenden GEMDOS-Aufrufen. Ein Problem liegt nun darin, daß diese Routinen andere Bezeichnungen und teilweise auch eine andere Parameterreihenfolge haben. Um nun die Quellen der Programme unverändert auf andere Rechner übernehmen zu können, müssen wir folgende Forderungen stellen:

a) Es werden nur Routinen benutzt, die auf allen Rechnern zur Verfügung stehen.

b) Es werden nur die Bezeichner der Routinen eines Rechners bzw. eines Betriebssystems benutzt.

Der Forderung a) kann nachgekommen werden, da die Routinen im GEMDOS, MS-DOS und FlexOS größtenteils identisch sind. Alle Routinen, die man benötigt, findet man entweder in „osbind.h" oder „dosbind.h". Auf Routinen wie z.B. für Tastatureingabe oder Bildschirmausgabe kann man getrost verzichten, da man für diesen Fall die GEM-Routinen benutzen kann. Ein übriges tun die Compilerbibliotheken, sofern in diesen auf jedem Rechner die gleichen Routinen implementiert sind. Dies gilt zumindest für die ANSI-Compiler.

Um Forderung b) zu erfüllen, müssen wir uns entscheiden, ob wir die GEMDOS- oder DOS-Namen benutzen. Wegen der großen Verbreitung von GEM auf dem Atari ST haben wir uns für die GEMDOS-Namen entschieden. Dies bedeutet, daß wir zum Beispiel zum Lesen des Inhaltsverzeichnisses die Routinen „Fsfirst" und „Fsnext" statt „dos_sfirst" und „dos_snext" verwenden.

Da die Benutzung der Bezeichner „Fsfirst" und „Fsnext" aber beim Linken unter MS-DOS zu undefinierten Symbolen führen würden, müssen wir diese vorher umdefinieren. Aus diesem Grund lesen wir die Datei „gemdos.h" ein, die sämtliche Funktionen, die in allen Betriebssystemen vorkommen, so umdefiniert, daß daraus die entsprechenden

Bezeichner für das jeweilige Betriebssystem werden. Für die zwei Aufrufe „Dgetpath" (dos_gdir) und „Fseek" (dos_lseek) wurden in den entsprechenden Makros auch die Parameter vertauscht, da diese in ihrer Reihenfolge nicht übereinstimmen.

Sollten Sie dennoch der Meinung sein, lieber MS-DOS- statt GEMDOS-Aufrufe zu benutzen, weil Sie eher unter MS-DOS entwickeln, so können Sie dies ruhig tun. Auch ein Mischbetrieb mit GEMDOS- und MS-DOS-Bezeichnern ist möglich. In diesem Fall müssen aber die Namen und Parameter für GEMDOS umdefiniert werden. Zu diesem Zweck haben wir die Datei „msdos.h" analog zu „gemdos.h" geschrieben, die genau dies tut. Sie muß nur noch eingelesen werden. Dies müßte an der entsprechenden Stelle im „import.h" geschehen, also z.B. direkt vor dem Befehl

```
include <osbind.h>
```

Wir haben das hier nicht getan, weil das ständige Einlesen dieser Datei unter GEMDOS zusätzliche Zeit kosten würde. Außerdem benutzen die auf der Begleitdiskette implementierten Module alle die GEMDOS-Namen.

Nun wird bei ANSI-Compilern noch die Datei „stdlib.h" aufgenommen, in der unter anderem die Routinen „abs" und „labs" definiert sind. Für Nicht-ANSI-Compiler werden die Funktionen als Makros nachdefiniert.

Es folgt die Definition von GLOBAL, wie schon weiter oben erklärt wurde. Für den Laser C werden noch die beiden Stringfunktionen „strchr" und „strrchr" auf „index" und „rindex" abgebildet.

Für den High C-Compiler von Metaware wird das im „stdio.h" definierte Makro NULL durch eine eigene Definition ersetzt, die für alle Compiler funktioniert. Würde man dies nicht tun, so würde man unberechtigterweise Fehlermeldungen bekommen.

Die beiden Makros „max" und „min" werden definiert, falls dies nicht bereits der Fall ist.

Das Makro „odd" ist TRUE, wenn eine Zahl ungerade ist, FALSE sonst.

Zum Schluß folgen noch Definitionen für Pascal- und Modula-Programmierer. Falls diese lieber ein „and" statt einem „&&" schreiben möchten oder „begin" statt einem „{", sollten sie vor dem Einlesen der Datei „import.h" oder am Anfang dieser Datei das Makro

```
#define PASCAL_DEF
```

bzw.

```
#define MODULA_DEF
```

aufnehmen. Man könnte dies bei den meisten Compilern auch von außen definieren, etwa durch die Compiler-Option „-D", die bei den meisten Compilern möglich ist, also

```
-DPASCAL_DEF
```

Falls die Makros nicht definiert sind, werden die Pascal- und Modula-Definitionen nicht benutzt. So spart man Zeit und Speicherplatz für den Präprozessor.

```
/*****************************************************************/
/*                                                               */
/* EXPORT.H                                                      */
/* Datum: 06/03/89                                               */
/*                                                               */
/*****************************************************************/

#ifndef __EXPORT__
#define __EXPORT__

#ifdef GLOBAL
#undef GLOBAL
#endif

#define GLOBAL

#endif /* __EXPORT __ */
```

Die Datei „export.h" besteht nur aus dem Umdefinieren des Schlüsselwortes GLOBAL, welches nun als leer definiert wird. Auf diese Weise wird Speicherplatz für Variablen angelegt, wie schon in 6.2 beschrieben wurde.

### 6.5.1 Modul GEMAIN

Wir beginnen mit einem einfachen Modul, welches nur eine einzige Aufgabe hat. Es stellt die Schnittstelle zwischen der C Run Time Library und unserer Applikation dar. Wegen der Kürze der Schnittstelle wollen wir diese hier komplett auflisten:

```
#if GEM & XGEM

GLOBAL WORD GEMAIN _((VOID));

#else

GLOBAL WORD main _((INT argc, BYTE *argv []));

#endif
```

Normalerweise heißt die Funktion eines C-Programms, die zuerst angesprungen wird, „main". Wie oben zu ersehen ist, würde dies bei der FlexOS-Implementierung unter X/GEM schiefgehen. Dort benutzt die X/GEM-Library das Symbol „main", um eigene Initialisierungen vor dem Hauptprogramm durchzuführen. Erst dann wird GEMAIN aufgerufen, das Hauptprogramm einer X/GEM-Applikation. Wie außerdem zu sehen ist, besitzt es keine Parameter.

Der C-Quelltext des Moduls GEMAIN soll hier auch abgedruckt werden, da er sehr kurz ist und zum Verständnis beiträgt.

```
/*****************************************************************/
/*                                                               */
/* Modul: GEMAIN.C                                               */
/* Datum: 27/05/89                                               */
/*                                                               */
/*****************************************************************/

#include "import.h"
#include "global.h"

#include "initerm.h"
#include "event.h"

#include "export.h"
#include "gemain.h"


/****** DEFINES ************************************************/

/****** TYPES **************************************************/

/****** VARIABLES **********************************************/

#if GEMDOS
#if MW_C
LONG _stksize = 12288;      /* 12 KBytes Stack für Mark Williams C */
#endif
#endif

#if MSDOS
#if TURBO_C
UWORD _stklen = 12288;      /* 12 KBytes Stack für Turbo C */
#endif
#endif
```

```
/****** FUNCTIONS ***************************************************/

#if GEM & XGEM

GLOBAL WORD GEMAIN ()
{
  if (init_initerm (0, NULL)) hndl_events ();
  term_initerm ();
  return (0);
} /* GEMAIN */

#else

/*******************************************************************/

GLOBAL WORD main (argc, argv)
INT  argc;
BYTE *argv [];

{
  if (init_initerm (argc, argv)) hndl_events ();
  term_initerm ();
  return (0);
} /* main */

#endif
```

Zunächst werden die benötigten Headerdateien eingelesen. Danach wird jeweils für den Mark Williams C-Compiler unter GEMDOS und den Turbo C-Compiler unter MS-DOS die Stackgröße festgelegt. Dies geschieht hier über Variablen, die vom Startup-Code der beiden Compilerbibliotheken erkannt werden.

Das eigentliche Hauptprogramm beginnt mit der Initialisierung aller Module über einen Aufruf von „init_initerm". Dabei werden die Argumente „argc" und „argv" übergeben. Unter X/GEM sind diese nicht verfügbar, so daß dort keine Argumente (0, NULL) übergeben werden. Dies ist aber nicht weiter tragisch, da dann die Argumente über „shel_read" gelesen werden.

Ist die Initialisierung in Ordnung, dann wird die Multi-Event-Schleife über den Aufruf „hndl_events" (im Modul EVENT) aufgerufen, in der alle Ereignisse abgefangen werden. Nach Terminierung dieser Funktion werden über „term_initerm" alle Module terminiert. Am Ende wird noch ein

```
return (0)
```

aufgerufen, damit der Aufrufer keine Fehlermeldung ausgibt.

Das Hauptmodul GEMAIN ist unabhängig von der eigentlichen Aufgabe der Applikation, sofern nur die Aufrufe „init_initerm", „term_initerm" und „hndl_events" mit entsprechenden Parametern existieren und dazugebunden werden.

### 6.5.2 Modul GLOBAL

GLOBAL ist das zentrale Modul der gesamten Applikation. Es bietet eine Menge vordefinierter Konstanten, Typen und Variablen an, die garantiert in jeder GEM-Applikation immer wieder auftreten. Dazu gibt es eine Menge von Funktionen, die von jeder Applikation mehr oder weniger gebraucht werden. Das richtige Benutzen dieses Moduls bietet außerdem die Garantie für Kompatibilität mit allen GEM-Versionen.

Sollten Sie in ihrer Applikation weitere Routinen benötigen, die „global" zu all Ihren Modulen sein sollen, so sollten Sie das Modul GLOBAL nicht erweitern, sondern ein Modul schreiben, welches oberhalb von GLOBAL steht, d.h. Routinen und Variablen aus GLOBAL benutzt, aber applikationsspezifisch ist. Dann bleibt GLOBAL applikationsunabhängig, und bei einer neuen Version dieses Moduls bzw. einer neuen Auflage dieses Buches muß nur das Modul GLOBAL ausgetauscht werden, ohne daß Modifikationen an anderen Modulen vorgenommen werden müssen (Aufwärtskompatibilität).

Die Konstanten und Typen in GLOBAL.H sind so gut kommentiert, daß eine weitere Erklärung hier überflüssig ist. Das gleiche gilt für die definierten Typen. Es werden hier nur noch die globalen Variablen sowie alle Funktionen erklärt. Die Funktionen werden jeweils in ANSI-Schreibweise angegeben.

a) Globale Variablen

```
#if DR_C | LASER_C | MW_C
EXTERN WORD gl_apid;
#else
GLOBAL WORD gl_apid;
#endif
```

Die globale Applikationsnummer wird von fast allen Aufrufen an die Applikations-Bibliothek benötigt. Je nach Compiler befindet sie sich in der AES-Bibliothek oder muß im Hauptprogramm definiert werden.

```
#if GEM & (GEM1 | GEM2 | GEM3)
GLOBAL WORD contrl [12];
GLOBAL WORD intin [256];
GLOBAL WORD ptsin [256];
GLOBAL WORD intout [256];
GLOBAL WORD ptsout [256];
#endif
```

Die Felder für die Parameter von VDI-Aufrufen werden bei X/GEM nicht benötigt. Bei einigen Compilern/Betriebssystemen müssen sie im Hauptprogramm definiert werden, bei anderen sind sie in der VDI-Bibliothek definiert und können damit als EXTERN definiert werden.

```
GLOBAL WORD gl_wchar;
GLOBAL WORD gl_hchar;
GLOBAL WORD gl_wbox;
GLOBAL WORD gl_hbox;
GLOBAL WORD gl_wattr;
GLOBAL WORD gl_hattr;
```

Die sechs Variablen geben globale Attribute des Systemzeichensatzes an. Die Variablen, die auf „char" enden, beinhalten die Breite und Höhe eines Zeichens aus dem Systemzeichensatz. Diese Werte können beim Aufruf von „vst_height" benutzt werden, um ein Zeichen aus dem (großen) Systemzeichensatz darzustellen. Dabei gilt die Höhe eines Zeichens nur bis zur Baseline. Bei einem 8x16-Zeichensatz wären die Werte z.B. 7x13.

Die Variablen, die auf „box" enden, geben die Breite und Höhe einer Box an, in die ein Zeichen des Systemzeichensatzes genau hineinpaßt. Bei einem 8x16-Zeichensatz wären dies die Werte 8x16. Die Anzahl der Pixel in X-Richtung und Anzahl der Pixel in Y-Richtung dividiert durch diese beiden Werte ergeben die Anzahl der Spalten bzw. Zeilen auf dem Bildschirm, also z.B. 640 / 8 = 80 Spalten und 400 / 16 = 25 Zeilen.

Die Variablen, die auf „attr" enden, geben die Breite und Höhe einer Box an, welche groß genug ist, um ein Attribut eines Fensters zu umschließen. Attribute von Fenstern sind z.B. die Schließbox, die Vergrößerungsbox usw. Diese Werte werden benutzt, wenn es um die Darstellung des Scrollbalkens der Menüzeile im Fenster geht.

GLOBAL WORD colors;

Sie beinhaltet die Anzahl der Farben. Wenn der Wert dieser Variablen zwei ist, kann man z.B. bestimmte Muster für Monochromdarstellung benutzen, da nur zwei Farben existieren.

GLOBAL WORD phys_handle;

Der durch „graf_handle" ermittelte Wert der physikalischen Workstation.

GLOBAL WORD vdi_handle;

Der durch „open_vwork" ermittelte Wert der virtuellen Workstation für den Bildschirm. Ist noch keine Workstation geöffnet, so hat „vdi_handle" den gleichen Wert wie „phys_handle".

GLOBAL BOOLEAN vwork_open;

Die Variable ist TRUE, wenn die virtuelle Workstation geöffnet ist, FALSE sonst. Durch Abfrage dieser Variablen kann verhindert werden, daß sie zweimal geöffnet wird (siehe Funktion „open_vwork").

GLOBAL BOOLEAN deskacc;

Die Variable ist TRUE, wenn die laufende Applikation ein Deskaccessory ist, FALSE sonst. Dies gilt für alle Compiler, auch solche, die keine externe Variable „_app" in ihr Runtime-Startup aufgenommen haben. Letztere haben allerdings den Vorteil, daß die damit entwickelten Programme auch als Applikationen laufen können, obwohl deren Name auf ACC endet.

GLOBAL BOOLEAN acc_close;

Die Variable ist TRUE, wenn die letzte Meldung von „evnt_multi" ein AC_CLOSE war, FALSE sonst. Es ist wichtig, sich diesen Umstand zu merken, da die beim AC_CLOSE zu schließenden Fenster tatsächlich nicht über „wind_close" bzw. „wind_delete" geschlossen bzw. gelöscht werden dürfen. Tut man dies trotzdem, stürzt das GEM gnadenlos ab. Eine Anwendung dieser Variablen findet man in der Funktion „close_window" im Modul WINDOWS.

GLOBAL WORD menu_id;

Der durch „menu_register" ermittelte Wert des Menüeintrags eines Deskaccessories.

GLOBAL WORD class_desk;

Die Variable erhält den Wert der Klasse des Desktops. Sie ist entweder null (DESK), wenn es sich um einen eigenen Desktop handelt, oder eins (DESKWINDOW), wenn der Desktop in ein Fenster gelegt werden muß. Dies ist genau dann der Fall, wenn es sich um ein Accessory mit eigenem Desktop handelt oder wenn die Menüzeile nicht mehr auf den Bildschirm paßt (z.B. bei zu breiter Menüzeile in der niedrigen Auflösung des Atari ST).

GLOBAL RECT desk;

Dies ist das Rechteck, welches den Desktop beschreibt. Es ist üblicherweise so groß wie der physikalische Bildschirm, wobei vom oberen Rand noch die Höhe der Menüzeile abgezogen wird.

GLOBAL RECT clip;

Dadurch wird das Rechteck, welches zuletzt mit „set_clip" (s.u.) gesetzt wurde, beschrieben. Damit können Algorithmen, die Fensterinhalte zeichnen, optimiert werden (siehe „wi_draw" im Modul EDIT).

GLOBAL WORD hidden;

Die Variable zählt die Anzahl der Aufrufe von „hide_mouse". Die „hide_mouse" und „show_mouse" Aufrufe werden gestackt, so daß immer eine korrespondierende Anzahl von Aufrufen stattfinden muß. Dadurch wird verhindert, daß Funktionen, welche nicht wissen, ob die sie aufrufenden Funktionen ebenfalls die Maus verstecken, die Maus wieder zeigen, obwohl sie noch unsichtbar sein sollte. Der Anfangswert ist null.

GLOBAL WORD busy;

Die Variable zählt die Anzahl der Aufrufe von „busy_mouse". Die „busy_mouse" und „arrow_mouse" Aufrufe werden gestackt, so daß immer eine korrespondierende Anzahl von Aufrufen stattfinden muß. Dadurch wird verhindert, daß Funktionen, welche nicht wissen, ob die sie aufrufenden Funktionen die Maus ebenfalls als Biene bzw. Stundenglas darstellen, die Maus wieder als Pfeil zeigen, obwohl sie noch geschäftig sein sollte. Der Anfangswert ist null.

GLOBAL BOOLEAN done;

Die Variable ist TRUE, wenn das Programm beendet werden soll, FALSE sonst. Dies ist üblicherweise beim Wählen der Funktion „Ende" der Fall. Der Anfangswert ist FALSE.

GLOBAL BOOLEAN grow_shrink;

Die Variable ist TRUE, wenn beim Öffnen und Schließen von Fenstern und Dialogboxen ein sich ausdehnendes bzw. schrumpfendes Rechteck gezeichnet werden soll, FALSE sonst. In unserer Beispielapplikation kann der Wert der Variablen über eine Dialogbox eingestellt werden. Der Anfangswert ist TRUE.

GLOBAL BOOLEAN ring_bell;

Die Variable ist TRUE, wenn bei Fehlermeldungen die Glocke ertönen soll, FALSE sonst. In unserer Beispielapplikation kann der Wert der Variablen über eine Dialogbox eingestellt werden. Damit kann das eventuell störende Piepssignal abgeschaltet werden. Der Anfangswert ist TRUE.

GLOBAL WORD blinkrate;

Sie gibt die Anzahl an, wie oft ein Menüeintrag jeweils invertiert werden soll, bevor das Menü ausgeführt wird. Dies gilt für Menüzeilen im Fenster und Pop-Up-Menüs. In unserer Beispielapplikation kann der Wert der Variablen über eine Dialogbox eingestellt werden. Der Anfangswert ist drei.

GLOBAL STR128 cmd;

Sie bezeichnet den Namen des Kommandos, durch welches das Programm gestartet wurde, d.h. den Namen des Programms selbst, also z.B. SCRAP.PRG. Der Wert wird über ein „shel_read" ermittelt.

GLOBAL STR128 tail;

Sie beschreibt die Parameter, die an die Applikation übergeben wurden. Sie werden aus „argv" oder über „shel_read" ermittelt, falls „argv" nicht gesetzt ist. Werden mehrere Parameter an das Programm übergeben, so werden diese jeweils durch ein Leerzeichen getrennt. So können Parameter unserer Beispielapplikation sein:

SCRAP.IMG SCRAP.GEM SCRAP.TXT

Es handelt sich also um drei Dateien. Sie alle werden beim Starten der Applikation geöffnet (siehe „init_initerm" im Modul INITERM).

GLOBAL STR128 called_by;

Sie bezeichnet den Namen des aufrufenden Programms, falls dieses seinen Namen hinter die Parameter gehängt hat (siehe auch Kapitel 5.5). Dieses wird dann beim Aufruf des Menüpunktes „An ..." aufgerufen (siehe „term_initerm" im Modul INITERM).

GLOBAL STR128 app_name;

Der Name der Applikation, also z.B. SCRAPP.APP oder SCRAP.ACC, wird hier abgelegt.

GLOBAL STR128 app_path;

Der Pfad der Applikation mit abschließendem „\" wird hier abgelegt, also z.B.

D:\TMP\SCRAP\

falls eine Applikation vom Inhaltsverzeichnis \TMP\SCRAP in Laufwerk D: gestartet wurde. Mit Hilfe dieses Pfades können Dateien, die sich im selben Inhaltsverzeichnis befinden sollen wie die Applikation, leicht eingelesen werden. Nicht immer ist das aktuell eingestellte Laufwerk und der aktuell eingestellte Pfad derjenige, auf welchem sich die Applikation befindet. Dies umso mehr, wenn eine Anwendung angemeldet wurde, und von einem anderen Laufwerk das angemeldete Piktogramm angeklickt wurde. Durch Voranstellen der Variable „app_path" kann ein kompletter Pfadname für eine Datei gebildet werden.

GLOBAL WORD act_drv;

Die Variable enthält das aktuelle Laufwerk beim Starten der Applikation.

GLOBAL STR128 act_path;

Die Variable enthält das aktuelle Inhaltsverzeichnis beim Starten der Applikation. Sie enthält auch den Laufwerksnamen und endet immer mit einem „\". Sie kann also z.B. den Wert

H:\SCRAP\

annehmen. Falls ein Dateiname ohne Pfad an eine Applikation übergeben wird, sollte der Name mit Hilfe dieser Variablen ergänzt werden.

GLOBAL STR128 scrapdir;

Sie enthält das Inhaltsverzeichnis des GEM-Clipboards. Es kann benutzt werden, um die Dateinamen zu erfahren, die sich darauf befinden. Als Maske sollte ein SCRAP.* gewählt werden. Das Inhaltsverzeichnis heißt CLIPBRD und befindet sich beim Atari auf der obersten Ebene eines schnellen Laufwerks (sofern vorhanden). Bei MS-DOS (ab GEM/3) befindet es sich im Ordner GEMAPPS, so daß der komplette Pfad für Laufwerk C

C:\GEMAPPS\CLIPBRD\

lauten würde. Eine Beschreibung zum GEM-Clipboard wurde schon in Kapitel 5 gegeben.

```
GLOBAL LONGSTR fs_path;
GLOBAL STRING  fs_sel;
GLOBAL WORD    fs_button;
```

Die Variablen geben den von der Applikation zuletzt gewählten Zugriffspfad (fs_path), den zuletzt gewählten Dateinamen (fs_sel) und den zuletzt betätigten Knopf (fs_button) der Dateiauswahl-Box an. Ein Beispiel für die Benutzung dieser Variablen wird in der Funktion „select_file" weiter unten gegeben.

```
GLOBAL OBJECT *menu;
GLOBAL OBJECT *about;
GLOBAL OBJECT *desktop;
GLOBAL OBJECT *freetext;
```

Die vier Objektbäume kommen in praktisch jeder Applikation vor und sind deshalb global definiert. „menu" gibt die Menüzeile an, „about" die Dialogbox, welche durch das Menü „Über..." gewählt wird. „desktop" ist der eigene Desktop und „freetext" ist eine Dialogbox, in der sich alle Texte befinden, die zu keiner bestimmten Dialogbox gehören. Alle Variablen werden auf NULL initialisiert und müssen nicht unbedingt gesetzt werden. Existiert beispielsweise keine Menüzeile oder kein eigener Desktop, so bleiben die Variablen eben auf NULL. Das Programm reagiert trotzdem korrekt.

GLOBAL BYTE **alertmsg;

Die Variable zeigt auf die Zeiger aller Fehlermeldungen. Sie wird im Modul RESOURCE initialisiert, falls es überhaupt Fehlermeldungen gibt. Die Variable wird benötigt, um die Funktionen „error" und „note" zu implementieren. Sie greifen dann auf die Fehlermeldungen über diese Variablen zu.

b) Globale Funktionen

– Öffnen und Schließen von Arbeitsstationen

VOID open_vwork (VOID);

Öffnet eine virtuelle Arbeitsstation auf dem Bildschirm. Die Variablen „vdi_handle", „colors" sowie „gl_wchar" und „gl_hchar" werden gesetzt. Ist die Arbeitsstation schon offen, so passiert nichts.

VOID close_vwork (VOID);

Schließt eine virtuelle Arbeitsstation auf dem Bildschirm. Diese muß vorher mit „open_vwork" geöffnet worden sein. Nach dem Schließen wird das „vdi_handle" auf „phys_handle" gesetzt. Ist die Arbeitsstation bereits geschlossen, so passiert nichts.

WORD open_work (WORD device, DEVINFO *dev_info);

Öffnet eine Arbeitsstation, also z.B. einen Bildschirm, einen Plotter, einen Drucker, eine Metadatei, eine Kamera oder ein Grafiktablett.

open_work: Handle der Arbeitsstation oder 0, wenn das Öffnen nicht geklappt hat.
device : Gerätenummer, wobei gilt:

| | |
|---|---|
| SCREEN | = 1 |
| PLOTTER | = 11 |
| PRINTER | = 21 |
| METAFILE | = 31 |
| CAMERA | = 41 |
| TABLET | = 51 |

Selbstverständlich können auch andere Nummern benutzt werden, falls diese einen sinnvollen Wert enthalten (z.B. 22 für den zweiten Drucker)
dev_info : Zeiger auf eine Struktur, die nach dem Aufruf bestimmte charakteristische Werte des Geräts enthält. Die Struktur ist folgendermaßen aufgebaut:

```
typedef struct
{
  LONG dev_w;  /* Breite des Ausgabegerätes in Pixel */
  LONG dev_h;  /* Höhe des Ausgabegerätes in Pixel */
  WORD pix_w;  /* Breite eines Pixels in 1/1000 mm */
  WORD pix_h;  /* Höhe eines Pixels in 1/1000 mm */
} DEVINFO;
```

Ist der Bildsschirm das zu öffnende Gerät, so wird es virtuell geöffnet.

VOID close_work (WORD device, WORD out_handle);

Die vorher mit „open_work" geöffnete Arbeitsstation wird wieder geschlossen.

device : Gerätenummer (siehe „open_work")
out_handle : Handle, welches durch „open_work" geliefert wurde

– Mausfunktionen

VOID set_mouse (WORD number, MFORM *addr);

Die aktuelle Mausform wird gesetzt.

number: Nummer der Mausform (siehe AES.H)
addr : Zeiger auf eine benutzerdefinierte Mausform, falls „number" den Wert 255 (= USER_DEF) hat

Die Funktion merkt sich die aktuelle und die letzte Mausform.

VOID last_mouse (VOID);

Setzt die letzte Mausform, die vor der aktuellen Mausform gültig war. Damit kann z.B. die Routine für Pop-Up-Menüs die alte Mausform wiederherstellen, nachdem während der Abarbeitung des Pop-Up-Menüs auf einen Pfeil umgeschaltet wurde.

VOID hide_mouse (VOID);

Die Maus wird versteckt. Ist die Maus bereits versteckt, passiert nichts. Die Anzahl der Aufrufe wird gezählt, so daß ein zweimaliger Aufruf von „hide_mouse" die Variable „hidden" auf 2 setzt. Um die Maus wieder sichtbar zu machen, müssen entsprechend viele Aufrufe von „show_mouse" erfolgen.

VOID show_mouse (VOID);

Die Maus wird sichtbar gemacht, wenn sie genau einmal versteckt war. Ansonsten passiert nichts. Wurde sie öfter versteckt, so muß sie genauso oft sichtbar gemacht werden, damit sie erscheint.

VOID busy_mouse (VOID);

Die Maus nimmt eine Form an, die anzeigt, daß der Rechner beschäftigt ist. Beim Atari ST ist dies eine Biene (BUSY_BEE), bei allen anderen Rechnern ein Stundenglas (HOURGLASS). Hat die Maus bereits diese Form, so passiert nichts. Um die Maus wieder zum Pfeil zu machen, müssen entsprechend viele Aufrufe von „arrow_mouse" erfolgen.

VOID arrow_mouse (VOID);

Die Mausform wird zum Pfeil, wenn sie genau einmal eine Biene bzw. ein Stundenglas war. Ansonsten passiert nichts. Durch die Zählung der „busy_mouse"- bzw. „arrow_mouse"-Aufrufe wird gewährleistet, daß bei länger andauernden Operationen, die eine beschäftigte Maus zeigen und in welchen Funktionen aufgerufen werden, die dies ebenfalls tun, die Maus durchgehend eine „beschäftigte" Form annimmt. Ohne Zählung würde sie vorzeitig eine Pfeilform annehmen, obwohl die Operation noch nicht abgeschlossen ist.

– Objektfunktionen

VOID do_state (OBJECT *tree, WORD obj, UWORD state);

Die Komponente „ob_state" eines Objekts wird gesetzt.

tree : Zeiger auf einen Objektbaum
obj : Index des Objekts
state : Status, welcher gesetzt wird (siehe AES.H)

VOID undo_state (OBJECT *tree, WORD obj, UWORD state);

Die Komponente „ob_state" eines Objekts wird zurückgesetzt.

tree : Zeiger auf einen Objektbaum
obj : Index des Objekts
state : Status, welcher zurückgesetzt wird (siehe AES.H)

VOID flip_state (OBJECT *tree, WORD obj, UWORD state);

Die Komponente „ob_state" eines Objekts wird invertiert, also z.B. von SELECTED auf NOT SELECTED.

tree : Zeiger auf einen Objektbaum
obj : Index des Objekts
state : Status, welcher invertiert wird (siehe AES.H)

WORD find_state (OBJECT *tree, WORD obj, UWORD state);

Die Objektnummer eines Objektes mit einem bestimmten Status (z.B. SELECTED) wird geliefert.

find_state: Index des Objekts mit dem gesuchten Status. Wurde kein Objekt gefunden, so ist das Ergebnis NIL.

tree : Zeiger auf einen Objektbaum
obj : Index des Objekts, bei welchem die Suche begonnen wird
state : Status, nach welchem gesucht wird (siehe AES.H)

VOID do_flags (OBJECT *tree, WORD obj, UWORD flag);

Die Komponente „ob_flags" eines Objekts wird gesetzt.

tree : Zeiger auf einen Objektbaum
obj : Index des Objekts
flag : Flag, welches gesetzt wird (siehe AES.H)

VOID undo_flags (OBJECT *tree, WORD obj, UWORD flag);

Die Komponente „ob_flags" eines Objekts wird zurückgesetzt.

tree: Zeiger auf einen Objektbaum
obj : Index des Objekts
flag: Flag, welches zurückgesetzt wird (siehe AES.H)

VOID flip_flags (OBJECT *tree, WORD obj, UWORD flag);

Die Komponente „ob_flags" eines Objekts wird invertiert, also z.B. von SELECTABLE auf NON SELECTABLE.

tree : Zeiger auf einen Objektbaum
obj : Index des Objekts
flag : Flag, welches invertiert wird (siehe AES.H)

WORD find_flags (OBJECT *tree, WORD obj, UWORD flag);

Die Objektnummer eines Objektes mit einem bestimmten Flag (z.B. SELECTABLE) wird geliefert.

find_flags: Index des Objekts mit dem gesuchten Flag. Wurde kein Objekt gefunden, so ist das Ergebnis NIL.

tree : Zeiger auf einen Objektbaum
obj : Index des Objekts, bei welchem die Suche begonnen wird
flag : Flag, nach welchem gesucht wird (siehe AES.H)

VOID set_ptext (OBJECT *tree, WORD obj, BYTE *s);

Der Wert der Zeichenkette eines Objektes vom Typ G_TEXT, G_BOXTEXT, G_FTEXT, G_FBOXTEXT kann gesetzt werden. Ist die übergebene Zeichenkette länger als die im Objektbaum reservierte Länge, so wird sie automatisch auf die maximale Länge verkürzt.

tree : Zeiger auf einen Objektbaum
obj : Index des Objekts
s : Zeiger auf die Zeichenkette, die im Objekt gesetzt werden soll

VOID get_ptext (OBJECT *tree, WORD obj, BYTE *s);

Der Wert der Zeichenkette eines Objektes vom Typ G_TEXT, G_BOXTEXT, G_FTEXT, G_FBOXTEXT kann gelesen werden.

tree : Zeiger auf einen Objektbaum
obj : Index des Objekts
s : Zeiger auf die Zeichenkette, die aus dem Objekt gelesen werden soll

VOID objc_rect (OBJECT *tree, WORD obj, RECT *rect);

Die aktuellen Koordinaten des ein Objekt umhüllenden Rechtecks können bestimmt werden. Das Ergebnis bezieht sich auf den Koordinatenursprung in der linken oberen Ecke des Bildschirms (x = 0, y = 0).

tree : Zeiger auf einen Objektbaum
obj : Index des Objekts
rect : Zeiger auf das resultierende Rechteck

VOID trans_gimage (OBJECT *tree, WORD obj);

Die Icons und Bit-Images eines Objektbaums können in das gerätespezifische Format umgewandelt werden. Dazu wird die VDI-Funktion „vr_trnfm" benutzt.

tree : Zeiger auf einen Objektbaum
obj : Index des Objekts

Ein Beispiel zu dieser Funktion findet sich im Modul RESOURCE, wo alle Objektbäume durchsucht werden, um Icons und Bit-Images zu transformieren. Der Aufruf ist nur dann nötig, wenn das gerätespezifische Format nicht dem Standardformat entspricht. Auf dem Atari ST müßten die Piktogramme nicht umgewandelt werden, aber es ist auch nicht falsch, es zu tun.

– Dialog-Funktionen

```
BOOLEAN background (OBJECT *tree, WORD obj, BOOLEAN get,
                    MFDB *screen, MFDB *buffer);
```

Der Hintergrund einer Dialogbox oder eines Menüs kann gerettet oder wieder restauriert werden. Der dafür zu verwendende Arbeitsspeicher wird automatisch vom System angefordert. Die Beschreibungen (MFDB's) für Bildschirm und Puffer werden automatisch eingetragen, lediglich der Speicherplatz dafür muß vom Aufrufer in Form zweier MFDB's zur Verfügung gestellt werden.

background: TRUE, wenn der Speicherplatz ausgereicht hat, um den Hintergrund zu retten, FALSE sonst
tree : Zeiger auf einen Objektbaum
obj : Index des Objekts
get : TRUE, wenn der Hintergrund geholt werden soll, FALSE, wenn er restauriert werden soll
screen : Zeiger auf die Beschreibung des Bildschirms
buffer : Zeiger auf die Beschreibung des Puffers

Bei allen Dialogboxen, Menüs und Pop-Up-Menüs werden alle Attribute berücksichtigt. Das bedeutet, daß auch der Hintergrund von eventuell vorhandenen, nach außen gehenden Rändern gerettet wird. Das gleiche gilt für eine eventuelle Schattierung der Objekte und für den Status OUTLINED. Natürlich darf auch eine beliebige Kombination davon auftreten. Die zu rettende Box wird auf eine Wortgrenze gebracht, um eine schnellere Arbeitsweise zu garantieren.

Hat der Rechner nicht mehr genügend Speicherplatz, um den Hintergrund abzulegen, so wird beim Restaurieren wenigstens ein „form_dial" mit FMD_FINISH aufgerufen, damit der Screen-Manager der Applikation die Nachricht sendet, Teile des Bildschirms neu aufzubauen. In diesem Fall wird das Funktionsergebnis auch FALSE sein.

```
BOOLEAN open_dial (OBJECT *tree, BOOLEAN grow, RECT *size,
                   MFDB *screen, MFDB *buffer);
```

Die Funktion bietet die Einleitung zum Öffnen und Behandeln einer Dialogbox. Optional kann ein sich ausdehnendes Rechteck gezeichnet werden und bestimmt werden, ob der Hintergrund gerettet werden soll. Die Beschreibungen (MFDB's) für Bildschirm und Puffer werden automatisch eingetragen, lediglich der Speicherplatz dafür muß vom Aufrufer in Form zweier MFDB's zur Verfügung gestellt werden.

open_dial: TRUE, wenn der Hintergrund gerettet werden sollte und auch konnte, FALSE sonst
tree : Zeiger auf einen Objektbaum
grow : TRUE, wenn ein sich ausdehnendes Rechteck gezeichnet werden soll, FALSE sonst
size : Zeiger auf die Beschreibung des sich ausdehnenden Ausgangsrechtecks oder NULL, wenn sich das Rechteck von der Bildschirmmitte ausdehnen soll
screen : Zeiger auf die Beschreibung des Bildschirms oder NULL, wenn der Hintergrund nicht gerettet werden soll
buffer : Zeiger auf die Beschreibung des Puffers oder NULL, wenn der Hintergrund nicht gerettet werden soll

Beim Abarbeiten eines Dialogs sollte die Maus ein Pfeil sein. Dieser wird hier gesetzt. Nach Beenden des Dialogs über „close_dial" wird wieder die vorherige Form angenommen.

Das sich ausdehnende Rechteck wird nur gezeichnet, wenn zusätzlich zum angegebenen Parameter die globale Variable „grow_shrink" TRUE ist.

Für X/GEM tritt das Retten des Bildschirmhintergrundes unabhängig von den Parametern nicht in Kraft. Der Grund liegt darin zu suchen, daß jeder Prozeß seinen eigenen abgeschlossenen Desktop hat. Dieser hat nur die Größe der Vereinigung aller in diesem Prozeß dargestellten und offenen Fenster. Würde statt eines Aufrufs von „form_dial" mit FMD_START der Hintergrund gerettet werden, so kann es passieren, daß dieser nicht vollständig auf dem Bildschirm erscheint, da alle aus dem Desktop herausragenden Teile geclippt werden.

```
BOOLEAN close_dial (OBJECT *tree, BOOLEAN shrink, RECT *size,
                    MFDB *screen, MFDB *buffer);
```

Die Funktion bietet das Beenden der Behandlung eines Dialoges. Optional kann ein sich zusammenziehendes Rechteck gezeichnet werden und bestimmt werden, ob der Bildschirm aus dem geretteten Hintergrund restauriert werden soll. Die Beschreibungen (MFDB's) für Bildschirm und Puffer werden automatisch eingetragen, lediglich der Speicherplatz dafür muß vom Aufrufer in Form zweier MFDB's zur Verfügung gestellt werden.

close_dial: TRUE, wenn der Hintergrund gerettet werden sollte und auch konnte, FALSE sonst
tree : Zeiger auf einen Objektbaum
shrink : TRUE, wenn ein sich zusammenziehendes Rechteck gezeichnet werden soll, FALSE sonst
size : Zeiger auf die Beschreibung des sich zusammenziehenden Endrechtecks oder NULL, wenn sich das Rechteck in die Bildschirmmitte zusammenziehen soll
screen : Zeiger auf die Beschreibung des Bildschirms oder NULL, wenn der Hintergrund nicht gerettet werden sollte
buffer : Zeiger auf die Beschreibung des Puffers oder NULL, wenn der Hintergrund nicht gerettet werden sollte

Das sich zusammenziehende Rechteck wird nur gezeichnet, wenn zusätzlich zum angegebenen Parameter die globale Variable „grow_shrink" TRUE ist.

Ist nach dem Aufruf von „close_dial" der Funktionswert FALSE, so muß genau dann der Hintergrund selbst restauriert werden, wenn dies durch die Meldungen des Screen-Managers nicht geschehen kann. Wird beispielsweise nach dem Öffnen einer Dialogbox durch Anwählen eines Knopfes eine weitere Dialogbox geöffnet (Unterdialog) und der Hintergrund konnte nicht gerettet werden, so muß nach dem Verlassen des zweiten Dialogs die erste Dialogbox wieder gezeichnet werden. Ein Beispiel ist im Modul MENU zu finden.

```
WORD hndl_dial (OBJECT *tree, WORD def, BOOLEAN grow_shrink,
                BOOLEAN save_back, RECT *size, BOOLEAN *ok);
```

Damit kann ein kompletter Dialog abgearbeitet werden, wenn er keine Unterdialoge (s.o.) enthält. Er bietet im Prinzip die Zusammenfassung von „open_dial", „form_do" und „close_dial". Als Funktionswert wird die Objektnummer des Knopfes zurückgegeben, mit dem der Dialog beendet wurde.

hndl_dial : Die Objektnummer des Knopfes, mit dem der Dialog verlassen wurde
tree : Zeiger auf einen Objektbaum
def : Default-Edit-Objekt, auf dem sich der Cursor befinden soll
grow_shrink: TRUE, wenn ein sich ausdehnendes und zusammenziehendes Rechteck gezeichnet werden soll, FALSE sonst

save_back : TRUE, wenn der Hintergrund gerettet werden soll, FALSE sonst
size : Beschreibung des Ausgangs- bzw. Endrechtecks für das Ausdehnen bzw. Schrumpfen oder NULL, wenn der Ausgangs- bzw. Endpunkt die Bildschirmmitte ist
ok : Zeiger auf eine Variable, die TRUE liefert, wenn der Hintergrund gerettet werden sollte und auch konnte, FALSE sonst

VOID blink (OBJECT *tree, WORD obj, WORD blinkrate);

Ein Objekt eines Objektbaumes wird zum Blinken veranlaßt. Dies kann z.B. bei Pop-Up-Menüs oder Menüzeilen im Fenster benutzt werden, um ein Macintosh-ähnliches Gefühl zu erzeugen.

tree : Zeiger auf einen Objektbaum
obj : Index des Objekts
blinkrate : Anzahl der Blinkwiederholungen

```
WORD popup_menu (OBJECT *tree, WORD obj, WORD x, WORD y,
                 WORD center_obj, BOOLEAN relative, WORD bmsk);
```

Hiermit wird die Verwaltung eines kompletten Pop-Up-Menüs übernommen. Es gibt mehrere Möglichkeiten, die Abarbeitung zu beeinflussen. Zunächst muß ein Objektbaum und ein Startobjekt angegeben werden. Außerdem können die X- und Y-Koordinaten angegeben werden, wo das Pop-Up-Menü genau erscheinen soll. Es ist aber auch möglich, das Pop-Up-Menü relativ zur Mauspostion anzeigen zu lassen, was ja eigentlich der Sinn eines Pop-Up-Menüs ist (kurze Mauswege). Schließlich kann man noch wählen, ob ein Objekt des Pop-Up-Menüs sich mit seinem Zentrum genau an der Mausposition befinden soll. Für das Verlassen des Pop-Up-Menüs kann der Mausknopf angegeben werden.

popup_menu: Index des Objekts, mit dem das Pop-Up-Menü beendet wurde
tree : Zeiger auf einen Objektbaum
obj : Index des Objekts (Pop-Up-Menüs)
x : X-Offset des Pop-Up-Menüs (absolut oder relativ)
y : Y-Offset des Pop-Up-Menüs (absolut oder relativ)
center_obj : Objekt, in dessen Zentrum die Maus erscheint oder NIL, wenn dieser Umstand nicht gewünscht wird
relative : TRUE, wenn sich das Pop-Up-Menü relativ zur Maus befinden soll, FALSE sonst.
bmsk : Maske für den Mausknopf, der das Pop-Up-Menü beenden soll. Das niederwertigste Bit gibt den am weitesten links stehenden Mausknopf an, usw.

Es bietet sich an, mehrere Pop-Up-Menüs in einem Objektbaum zusammenzufassen. „obj" ist dann das Vaterobjekt des jeweiligen Pop-Up-Menüs (siehe auch SCRAP.RSC).

Es werden nur jeweils die Einträge eines Pop-Up-Menüs invertiert, die den Status SELECTABLE haben. Sie müssen selbstverständlich auch nicht DISABLED sein. Dadurch ist es möglich, weitere Objekte wie Überschriften und erklärenden Text in ein Pop-Up-Menü hineinzubringen. Diese können dann nicht angewählt werden.

Ein Pop-Up-Menü wird immer auf dem Bildschirm dargestellt. Sollte es an irgendeinem Rand aus diesem Bereich herausragen, so werden die Koordinaten entsprechend korrigiert.

Ein Pop-Up-Menü kann auf zwei Arten beendet werden: entweder durch Loslassen des in „bmsk" angegebenen Mausknopfes oder durch dessen Drücken (siehe auch Kapitel 5). Das hängt davon ab, wie man das Pop-Up-Menü aufgerufen hat. Wurde der Mausknopf gedrückt und nicht losgelassen, so kann man das Pop-Up-Menü durch Loslassen des Mausknopfes beenden (Macintosh-like). Wurde der Mausknopf gedrückt und sofort wieder losgelassen, so muß man auf das entsprechende Objekt noch einmal klicken (GEM-like).

Beim Verlassen des Pop-Up-Menüs blinkt das zuletzt angewählte Objekt, je nachdem, wie die globale Variable „blinkrate" eingestellt wurde.

```
BOOLEAN is_menu_key (OBJECT *menu, MKINFO *mk,
                     WORD *title, WORD *item);
```

Der Titel und der Menüeintrag eines zu einer Taste gehörigen Menüs kann bestimmt werden. Die Tastenkürzel müssen sich am rechten Ende des Menüeintrags befinden (siehe auch Abb. 5.7). Die Taste und der Status der Shift-, Control- und Alternate-Tasten werden in einer Struktur MKINFO übergeben.

| | | |
|---|---|---|
| is_menu_key | : | TRUE, wenn die Taste auf ein Menü zutrifft, FALSE sonst |
| menu | : | Zeiger auf einen Menübaum |
| mk | : | Zeiger auf eine Struktur, die die aktuell betätigte Taste beschreibt |
| title | : | Zeiger auf den Index des Titels, falls „is_menu_key" TRUE ist |
| item | : | Zeiger auf den Index des Menüeintrags, falls „is_menu_key" TRUE ist |

Falls kein Menü gefunden wurde, ist „title" und „item" jeweils NIL. Es werden nur Einträge der Menüzeile untersucht, die den Typ G_STRING haben, also keine Bit-Images etc. Zum Erkennen der Tasten dient eine Tabelle, welche in GLOBAL.C definiert ist. Sie enthält alle Codes für Control- und Alternate-Kombinationen sowie Funktionstastencodes.

Die Menüeinträge können auch nach Start des Programms geändert werden. Solange sich die Kürzel am Ende eines Menüeintrags befinden, werden diese erkannt. Als Standardsymbole für die Kürzel werden die aus Abbildung 5.7 gezeigten Symbole benutzt (CTRL_CHAR, ALT_CHAR, SHIFT_CHAR, FUNC_CHAR in GLOBAL.C).

– Rechteckfunktionen

Die Rechteckfunktionen gehen über das hinaus, was in einigen Compilerbibliotheken angeboten wird. Das Problem ist, daß wichtige Funktionen wie z.B. „rc_intersect" nicht in jeder Bibliothek vertreten sind. Somit haben wir sie implementiert und sind damit unabhängig von den Compilerbibliotheken.

BOOLEAN rc_equal (CONST RECT *p1, CONST RECT *p2);

Zwei Rechtecke können auf Gleichheit getestet werden.

rc_equal: TRUE, wenn die Rechtecke gleich sind, FALSE sonst
p1 : Zeiger auf Rechteck 1
p2 : Zeiger auf Rechteck 2

VOID rc_copy (CONST RECT *ps, RECT *pd);

Die Beschreibung eines Rechtecks wird in ein anderes kopiert.

ps: Zeiger auf das Quellrechteck
pd: Zeiger auf das Zielrechteck

VOID rc_union (CONST RECT *p1, RECT *p2);

Zwei Rechtecke werden vereinigt. Das Vereinigungsrechteck wird im zweiten Parameter (p2) abgelegt.

p1: Zeiger auf das erste Rechteck
p2: Zeiger auf das zweite Rechteck (Vereinigungsrechteck)

Ist die Breite oder Höhe des zweiten Rechtecks null, so ist das Ergebnis die Beschreibung des ersten Rechtecks.

BOOLEAN rc_intersect (CONST RECT *p1, RECT *p2);

Zwei Rechtecke werden geschnitten. Das Schnittrechteck wird im zweiten Parameter (p2) abgelegt.

rc_intersect: TRUE, wenn das Schnittrechteck nicht leer ist, FALSE sonst
p1 : Zeiger auf das erste Rechteck
p2 : Zeiger auf das zweite Rechteck (Schnittrechteck)

BOOLEAN inside (WORD x, WORD y, CONST RECT *r);

Testet, ob ein Punkt in einem Rechteck liegt.

inside: TRUE, falls der Punkt innerhalb des Rechtecks liegt, FALSE sonst
x : X-Koordinate des Punkts
y : Y-Koordinate des Punkts
r : Zeiger auf das Rechteck, in welchem der Punkt liegen soll

VOID rect2array (CONST RECT *rect, WORD *array);

Wandelt eine Rechteckbeschreibung in eine Feldbeschreibung um. Die Rechteckbeschreibung besteht aus X- und Y-Koordinate sowie Breite und Höhe. Die Feldbeschreibung besteht aus den zwei diagonal gegenüberliegenden Eckpunkten des Rechtecks. Für Funktionen des AES benötigt man meist eine Rechteckbeschreibung, für solche des VDI meist eine Eckpunktbeschreibung.

rect : Zeiger auf das Rechteck
array : Zeiger auf die Feldbeschreibung

VOID array2rect (CONST WORD *array, RECT *rect);

Gegenstück zu „rect2array".

array : Zeiger auf die Feldbeschreibung
rect : Zeiger auf das Rechteck

VOID xywh2array (WORD x, WORD y, WORD w, WORD h, WORD *array);

Wie „rect2array", nur daß statt der Rechteckbeschreibung in einer Struktur vier einzelne Werte übergeben werden.

x : X-Koordinate des Rechtecks
y : Y-Koordinate des Rechtecks
w : Breite des Rechtecks
h : Höhe des Rechtecks
array : Zeiger auf die Feldbeschreibung

```
VOID array2xywh (CONST WORD *array,
                 WORD *x, WORD *y, WORD *w, WORD *h);
```

Gegenstück zu „xywh2array".

array : Zeiger auf die Feldbeschreibung
x : Zeiger auf die X-Koordinate des Rechtecks
y : Zeiger auf die Y-Koordinate des Rechtecks
w : Zeiger auf die Breite des Rechtecks
h : Zeiger auf die Höhe des Rechtecks

VOID xywh2rect (WORD x, WORD y, WORD w, WORD h, RECT *rect);

Eine Rechteckbeschreibung wird mit den vier Werten X-, Y-Koodinate, Breite und Höhe gefüllt.

x : X-Koordinate des Rechtecks
y : Y-Koordinate des Rechtecks
w : Breite des Rechtecks
h : Höhe des Rechtecks
rect : Zeiger auf das Rechteck

VOID rect2xywh (CONST RECT *rect,
WORD *x, WORD *y, WORD *w, WORD *h);

Gegenstück zu „xywh2rect".

rect : Zeiger auf das Rechteck
x : Zeiger auf die X-Koordinate des Rechtecks
y : Zeiger auf die Y-Koordinate des Rechtecks
w : Zeiger auf die Breite des Rechtecks
h : Zeiger auf die Höhe des Rechtecks

VOID set_clip (BOOLEAN flag, CONST RECT *size);

Das Clipping-Rechteck für grafische Ausgaben über das VDI kann gesetzt oder zurückgesetzt werden.

flag: TRUE, wenn das Clipping gesetzt, FALSE wenn es zurückgesetzt (deaktiviert) werden soll
size: Zeiger auf das Rechteck, welches das Clipping beschreibt

Wird kein Rechteck angegeben (size == NULL), so wird der Desktop als Rechteck angenommen. Die zuletzt gesetzten Werte werden in der globalen Variablen „clip" gesichert.

VOID growbox (CONST RECT *st, CONST RECT *fin);

Ein sich ausdehnendes Rechteck kann gezeichnet werden. Falls die globale Variable „grow_shrink" auf FALSE steht, wird der Aufruf ignoriert.

st : Startrechteck (kleines Rechteck)
fin : Endrechteck (großes Rechteck)

Für GEM-Versionen, in welchen keine „graf_growbox"- bzw. „graf_shrinkbox"-Funktionen implementiert sind, werden diese mittels der „xgrf"-Library nachgebildet.

VOID shrinkbox (CONST RECT *fin, CONST RECT *st);

Gegenstück zu „growbox". Auch hier wird die globale Variable „grow_shrink" berücksichtigt.

fin : Endrechteck (kleines Rechteck)
st : Startrechteck (großes Rechteck)

– Fehlerbehandlung

VOID beep (VOID);

Gibt ein Piepszeichen aus, sofern die globale Variable „ring_bell" auf TRUE steht. Sonst wird der Aufruf ignoriert. Für Atari's GEM wird ein BIOS-Aufruf benutzt, bei allen anderen GEM-Versionen der VDI-Aufruf „v_sound". Das Piepszeichen wird auch ausgegeben, falls noch keine virtuelle Workstation geöffnet war, da das „phys_handle" benutzt wird.

WORD note (WORD button, WORD index, WORD helpinx, OBJECT *helptree);

Es kann eine Warnmeldung oder Notiz in Form einer Alertbox auf dem Bildschirm ausgegeben werden. Zusätzlich wird noch die Möglichkeit geboten, eine Hilfemeldung einzublenden, falls der Benutzer auf einen Knopf mit z.B. der Aufschrift „HILFE" drückt.

note : Nummer des Knopfes, mit dem die Alertbox verlassen wurde (normalerweise 1 bis 3)
button : Default-Button der Alertbox
index : Nummer der Alertbox, meist ein Bezeichner aus der Resource-Datei
helpinx : Nummer des Knopfes, bei dessen Betätigung eine Hilfemeldung eingeblendet werden soll, oder NIL
helptree : Baum, der die Hilfemeldung beherbergt, oder NULL

Möchte man dem Benutzer zu jeder Fehler- und Warnmeldung die Möglichkeit geben, eine Hilfemeldung einzublenden, so übergibt man in „helpinx" die Nummer des Knopfes, der die Aufschrift „HILFE" trägt. Bietet man keine Hilfe an, so kann man auch NIL übergeben. Falls der Benutzer auf „HILFE" klickt, so erscheint die unter „helptree" angegebene Dialogbox. Der Hintergrund wird automatisch gerettet. Nach Abarbeiten der Hilfe-Dialogbox wird die Warnmeldung wieder eingeblendet. Der Benutzer kann nun wieder „HILFE" wählen oder auch „OK", „ABBRUCH" und Ähnliches.

Ein Zeiger auf die Adressen der Warn- oder Fehlermeldungen, welche im RCS erstellt wurden, müssen vorher in der Variablen „alertmsg" untergebracht werden. Dies geschieht im Modul RESOURCE.

WORD error (WORD button, WORD index, WORD helpinx, OBJECT *helptree);

Es kann eine Fehlermeldung ausgegeben werden. Der Ablauf ist der gleiche wie bei „note", nur daß zusätzlich ein Piepszeichen ausgegeben wird, falls die globale Variable „ring_bell" auf TRUE gestellt ist.

error : Nummer des Knopfes, mit dem die Alertbox verlassen wurde (normalerweise 1 bis 3)
button : Default-Button der Alertbox
index : Nummer der Alertbox, meist ein Bezeichner aus der Resource-Datei
helpinx : Nummer des Knopfes, bei dessen Betätigung eine Hilfemeldung eingeblendet werden soll, oder NIL
helptree : Baum, der die Hilfemeldung beherbergt, oder NULL

– Speicherfunktionen

VOID *mem_alloc (LONG mem);

Der damit allokierte Speicher wird über eine Betriebssystemfunktion aufgerufen, nicht über eine Standard-C-Funktion, da sich nicht bei jedem Compiler mehr als 64KByte allokieren lassen.

mem_alloc: Zeiger auf den allokierten Speicherbereich oder NULL, wenn kein Speicher mehr frei war.
mem : Anzahl der zu allokierenden Bytes

BOOLEAN mem_free (VOID *memptr);

Der vorher mit „mem_alloc" allokierte Speicherplatz kann hiermit wieder freigegeben werden.

mem_free : TRUE, wenn die Freigabe geklappt hat, FALSE sonst
memptr : Zeiger auf den vorher durch „mem_alloc“ angeforderten Speicherplatz

LONG mem_avail (VOID);

Die Anzahl der noch freien Bytes des Hauptspeichers kann ermittelt werden.

mem_avail: Anzahl der freien Bytes des Hauptspeichers

VOID *mem_set (VOID *dest, WORD val, UWORD len);

Ein bestimmter Bereich des Speichers (maximal 64KByte) kann mit einem Wert gefüllt werden. Ist diese Funktion in einer Compiler-Bibliothek vorhanden, so wird sie benutzt, ansonsten hier implementiert.

mem_set : Zeiger auf den Zielspeicherbereich (dest).
dest : Zeiger auf den Speicherbereich, der gefüllt werden soll
val : Wert, mit dem der Speicherbereich gefüllt wird
len : Anzahl der zu füllenden Bytes

VOID *mem_move (VOID *dest, CONST VOID *src, UWORD len);

Ein Speicherbereich (maximal 64KByte) kann in einen anderen verschoben werden. Dabei werden Überlappungen automatisch berücksichtigt. Ist diese Funktion in einer Compiler-Bibliothek vorhanden, so wird sie benutzt, ansonsten hier implementiert.

mem_move: Zeiger auf den Zielbereich (dest)
dest : Zeiger auf den Zielbereich
src : Zeiger auf den Ursprungsbereich
len : Anzahl der zu verschiebenden Bytes

VOID *mem_lset (VOID *dest, WORD val, ULONG len);

Wie „mem_set“, jedoch können maximal 4GByte gefüllt werden.

mem_set : Zeiger auf den Zielspeicherbereich (dest).
dest : Zeiger auf den Speicherbereich, der gefüllt werden soll
val : Wert, mit dem der Speicherbereich gefüllt wird
len : Anzahl der zu füllenden Bytes

VOID *mem_lmove (VOID *dest, CONST VOID *src, ULONG len);

Wie „mem_move", jedoch können maximal 4GByte verschoben werden.

mem_move: Zeiger auf den Zielbereich (dest)
dest : Zeiger auf den Zielbereich
src : Zeiger auf den Ursprungsbereich
len : Anzahl der zu verschiebenden Bytes

– Zeichenkettenfunktionen

BYTE *str_upper (BYTE *s);

Die Buchstaben einer Zeichenkette werden in Großbuchstaben umgewandelt. Nationale Sonderzeichen werden nicht berücksichtigt. Ist diese Funktion in einer Compiler-Bibliothek vorhanden, so wird sie benutzt, ansonsten hier implementiert.

str_upper : Zeiger auf die Zeichenkette (s)
s : Zeiger auf die Zeichenkette, die konvertiert werden soll

BYTE *str_lower (BYTE *s);

Die Buchstaben einer Zeichenkette werden in Kleinbuchstaben umgewandelt. Nationale Sonderzeichen werden nicht berücksichtigt. Ist diese Funktion in einer Compiler-Bibliothek vorhanden, so wird sie benutzt, ansonsten hier implementiert.

str_lower : Zeiger auf die Zeichenkette (s)
s : Zeiger auf die Zeichenkette, die konvertiert werden soll

– Mengenfunktionen

Mengen sind Datentypen, die eine bestimmte Anzahl von Objekten aufnehmen können. Sie repräsentieren genau das, was in der Mathematik auch als Menge bezeichnet wird – nur mit einem endlichen Vorrat an Elementen. In Programmiersprachen wie Pascal oder Modula gibt es solche Mengen, auch Sets genannt.

Ist die Anzahl der Elemente einer Menge null, so spricht man auch von einer leeren Menge. Auf Mengen können verschiedene Operationen angewendet werden. Intern werden sie durch einen Bitvektor realisiert. Dieser Bitvektor hat eine bestimmte Größe. Jedes Bit entspricht genau einem Element. In einem Langwort beispielsweise können genau 32 Elemente untergebracht werden. Wir werden größere Mengen benötigen und definieren eine Menge als eine Folge von 32 Langwörtern, d.h. eine Menge kann 32x32 = 1024 Elemente besitzen. Die folgenden Definitionen (siehe GLOBAL.H) spiegeln diesen Umstand wider:

```
#define SETSIZE     32
#define SETMAX      (SETSIZE * 32 - 1)

typedef ULONG SET [SETSIZE];
```

Der Vorteil einer Menge gegenüber herkömmlichen Variablen besteht darin, daß sehr schnell getestet werden kann, ob bestimmte Elemente in dieser Menge enthalten sind. Außerdem kann ein Element nur einmal in einer Menge vorkommen. Eine Anwendung für Mengen sind Objekte, die vom Benutzer in einem Fenster angewählt werden können. Jedes Objekt hat eine Nummer (von null aufsteigend). Maximal 1024 Objekte (0 – 1023) können in einer solchen Menge Platz finden. Zu den logischen Operatoren, die auf eine Menge angewendet werden können, zählen z.B. Durchschnitt, Vereinigung und Differenz.

Anwendungsbeispiele für Mengen finden sich in den Modulen CLIPBRD und DESKTOP sowie MENU.

VOID setcpy (SET set1, CONST SET set2);

Eine Menge wird in eine andere kopiert.

set1: Zeiger auf die Zielmenge
set2: Zeiger auf die Quellmenge

VOID setall (SET set);

Alle Elemente einer Menge werden gesetzt.

set: Zeiger auf die Menge, in der die Elemente gesetzt werden

VOID setclr (SET set);

Alle Elemente einer Menge werden gelöscht.

set: Zeiger auf die Menge, in der die Elemente gelöscht werden

VOID setnot (SET set);

Alle Elemente einer Menge werden negiert. Elemente, die vorher nicht in der Menge waren, befinden sich nun in der Menge. Elemente, die vorher in der Menge waren, befinden sich nun nicht mehr in der Menge.

set: Zeiger auf die Menge, in der die Elemente negiert werden

VOID setand (SET set1, CONST SET set2);

Zwei Mengen werden mit einer UND-Operation verknüpft (Durchschnitt). Das Ergebnis ist eine Menge, in der nur die Elemente vorkommen, die in beiden Mengen vorhanden waren. Das Ergebnis (=Zielmenge) findet sich in der ersten Menge.

set1: Zeiger auf erste Menge (= Zielmenge)
set2: Zeiger auf zweite Menge

VOID setor (SET set1, CONST SET set2);

Zwei Mengen werden mit einer ODER-Operation verknüpft (Vereinigung). Das Ergebnis ist eine Menge, in der die Elemente vorkommen, die in der einen oder der anderen Menge vorhanden waren. Das Ergebnis (=Zielmenge) findet sich in der ersten Menge.

set1: Zeiger auf erste Menge (= Zielmenge)
set2: Zeiger auf zweite Menge

VOID setxor (SET set1, CONST SET set2);

Zwei Mengen werden mit einer Exklusiv-ODER-Operation verknüpft (symmetrische Differenz). Das Ergebnis ist eine Menge, in der nur die Elemente vorkommen, die nicht in einer der beiden Mengen vorhanden waren. Das Ergebnis (=Zielmenge) findet sich in der ersten Menge.

set1: Zeiger auf erste Menge (= Zielmenge)
set2: Zeiger auf zweite Menge

VOID setincl (SET set, WORD elt);

Ein Element wird einer Menge hinzugefügt. Ist dieses Element bereits darin enthalten, so passiert nichts.

set: Zeiger auf die Menge, zu der ein Element hinzugefügt wird
elt: Element, welches hinzugefügt werden soll

VOID setexcl (SET set, WORD elt);

Ein Element wird aus einer Menge entfernt. War dieses Element nicht in der Menge enthalten, so passiert nichts.

set: Zeiger auf die Menge, aus der ein Element entfernt wird
elt: Element, welches entfernt werden soll

BOOLEAN setin (CONST SET set, WORD elt);

Es wird geprüft, ob ein Element in einer Menge ist oder nicht.

setin: TRUE, wenn das Element in der Menge ist, FALSE sonst
set : Zeiger auf die Menge, die geprüft werden soll
elt : Element, welches geprüft werden soll

BOOLEAN setcmp (CONST SET set1, CONST SET set2);

Zwei Mengen werden miteinander verglichen. Sind sie gleich, so wird TRUE zurückgegeben. Gibt man keine zweite Menge an (NULL), so wird mit der leeren Menge verglichen.

setcmp: TRUE, wenn beide Mengen gleich sind, FALSE sonst
set1 Zeiger auf die erste Menge :
set2 Zeiger auf die zweite Menge oder NULL, wenn mit der leeren Menge verglichen werden soll

WORD setcard (CONST SET set);

Die Anzahl der Elemente einer Menge wird berechnet.

setcard: Anzahl der Elemente der Menge
set : Zeiger auf die Menge, deren Elementanzahl berechnet wird

– Verschiedenes

VOID file_split (BYTE *fullname, WORD *drive, BYTE *path,
BYTE *filename, BYTE *ext);

proportionaler Zeichensatz

Ein Dateiname wird in seine Komponenten zerlegt. Existieren bestimmte Komponenten nicht (Laufwerk oder Pfad), so werden die aktuellen Werte eingesetzt.

fullname: Dateiname, der alle oder ein Teil der Komponenten enthalten kann
drive : Zeiger auf das Laufwerk, welches im Dateinamen angegeben wurde, oder auf das aktuelle Laufwerk, wenn keines angegeben wurde
path : Zeiger auf den Pfad mit abschließendem '\', welcher im Dateinamen angegeben wurde, oder den aktuellen Pfad, wenn keiner angegeben wurde
filename: Zeiger auf den Dateinamen
ext : Zeiger auf die Extension

Falls „drive" NULL ist, wird kein Laufwerk zurückgegeben. Falls „path" NULL ist, wird kein Pfad zurückgeliefert. Falls „filename" NULL ist, wird kein Dateiname zurückgeliefert. Falls „ext" NULL ist, wird die Extension im Dateinamen angegeben (falls eine existiert). Die Extension eines Dateinamens erhält man also z.B. in der Variablen „ext" durch einen Aufruf von

file_split (fullname, NULL, NULL, NULL, ext),

wenn „fullname" den gesamten Namen angibt.

BOOLEAN get_path (BYTE *path, WORD drive);

Der aktuelle Pfad eines Laufwerks kann geholt werden. Der Pfad ist immer vollständig, und zwar unabhängig vom Betriebssystem, d.h. mit führendem und abschließendem „\".

get_path: TRUE, wenn der Pfad zum Laufwerk existiert, FALSE sonst
path : Zeiger auf den Pfad
drive : Laufwerk, von dem der Pfad bestimmt werden soll
0 = aktuelles Laufwerk
1 = Laufwerk A
2 = Laufwerk B, usw.

BOOLEAN set_path (CONST BYTE *path);

Der aktuelle Pfad eines Laufwerks kann gesetzt werden. Der Pfad kann unabhängig vom Betriebssystem mit oder ohne abschließendem „\" sein.

set_path : TRUE, wenn der Pfad gesetzt werden konnte, FALSE sonst
path : Zeiger auf den Pfad, der gesetzt werden soll (mit oder ohne Laufwerksangabe)

BOOLEAN file_exist (CONST BYTE *filename);

Es kann geprüft werden, ob eine Datei existiert oder nicht, ohne auf diese zuzugreifen. Dies ist besonders im Multiuser-Betrieb günstig (Shared-Open nicht nötig).

file_exist: TRUE, wenn die Datei existiert, FALSE sonst
filename : Zeiger auf den Dateinamen

BOOLEAN path_exist (CONST BYTE *pathname);

Es kann geprüft werden, ob ein Pfad existiert oder nicht. Dabei ist es irrelevant, ob der abschließende „\" angegeben wird oder nicht.

path_exist: TRUE, wenn der Pfad existiert, FALSE sonst
pathname : Zeiger auf den Pfadnamen

```
BOOLEAN select_file (CONST BYTE *name, CONST BYTE *suffix,
                     BYTE *filename);
```

Mit Hilfe der Dateiauswahl-Box von GEM kann eine Datei gewählt werden. Dabei kann ein Standard-Namen (z.B. NAMENLOS.DOC) sowie ein Suffix angegeben werden (z.B. „*.DOC").

select_file: TRUE, wenn eine Datei ausgewählt und OK gewählt wurde, FALSE sonst
name : Zeiger auf einen Dateinamen, der als Default-Namen angeboten wird
suffix : Zeiger auf ein Suffix, welches als Default-Suffix angeboten wird
filename : Resultierender kompletter Dateiname

Bei dieser Funktion werden auch die globalen Variablen „fs_path", „fs_sel" und „fs_button" gesetzt. Ist der Parameter „suffix" nicht leer, so wird er in die Dateiauswahl-Box gesetzt, ansonsten wird das zuletzt eingestellte Suffix benutzt.

Ist der Parameter „name" nicht leer, so wird er in die Dateiauswahl-Box gesetzt, ansonsten wird der zuletzt gewählte Name nicht verändert.

Wird zwar der OK-Knopf gewählt, aber keine Datei spezifiziert, so hat dies dieselbe Wirkung wie das Wählen des Abbruch-Knopfes.

```
BOOLEAN init_global (INT argc, BYTE *argv [], BYTE *acc_menu,
                     WORD class);
```

Das Modul GLOBAL wird initialisiert. Dieser Aufruf muß erfolgen, bevor andere Funktionen des Moduls benutzt werden.

init_global: TRUE, wenn die Initialisierung geklappt hat, FALSE sonst
argc : „argc" aus der Parameterliste von „main"
argv : „argv" aus der Parameterliste von „main"
acc_menu : Zeiger auf die Zeichenkette, die in das Accessory-Menü geschrieben wird
class : Klasse des Desktops, entweder DESK oder DESKWINDOW (vor allem X/GEM)

Im einzelnen laufen folgende Schritte ab:

a) Die Applikation wird beim GEM angemeldet (appl_init). Dies geschieht unabhängig und korrekt für jede GEM-Version und jeden Compiler.

b) Die globalen Variablen werden vorbesetzt. Dazu gehört das Ermitteln des „phys_handle", der Zeichenbreiten und -höhen und der Desktop-Größe.

c) Das aktuelle Laufwerk und der aktuelle Zugriffspfad werden geholt.

d) Die Parameter aus der Kommandozeile werden ermittelt. Ist „argc" größer als eins, so werden „argc" und „argv" benutzt, andernfalls „shel_read". Falls mehrere Parameter existieren, werden diese hintereinander gehängt.

e) Aus der Kommandozeile wird der Name des aufrufenden Programms extrahiert, sofern dieser existiert.

f) Falls die Kommandozeile Kommata enthält, werden diese durch Leerzeichen ersetzt (siehe auch Kapitel 5.5).

g) Der Programmname (ohne Pfad) und der Pfadname der Applikation werden unabhängig von der GEM-Version ermittelt.

h) Es wird untersucht, ob es sich beim laufenden Programm um eine Applikation oder ein Accessory handelt. Dies geschieht unabhängig vom Betriebssystem, von der GEM-Version oder dem benutzten Compiler und funktioniert auch für Accessories, welche ein „rsrc_load" verwenden.

i) Falls es sich um ein Accessory handelt, wird das Accessory-Menü gesetzt. Außerdem wird der Desktop, falls vorhanden, in ein echtes Fenster gelegt. Handelt es sich nicht um ein Accessory, wird lediglich die Klasse des Desktops aus dem Parameter „class" übernommen.

j) Das Scrapdirectory wird geholt. Ist es noch nicht gesetzt, wird es nach dem in Kapitel 5 vorgestellten Algorithmus ermittelt und gesetzt.

BOOLEAN term_global (VOID);

Das Modul GLOBAL wird terminiert. Dieser Aufruf sollte der letzte eines Programms sein. Die virtuelle Workstation wird geschlossen, falls sie geöffnet war, und die Applikation beim GEM abgemeldet.

term_global: TRUE, wenn die Terminierung geklappt hat, FALSE sonst

### 6.5.3 Modul WINDOWS

Das Modul WINDOWS bietet einen flexiblen Window-Manager, der oberhalb des GEM-Window-Managers anzusetzen ist. Es vereinfacht die Fensterhandhabung durch Zusammenfassen von GEM-typischen Sequenzen, ohne dabei an Flexibilität einzubüßen. Ein

Fenster kann man sich als Objekt vorstellen, welches auf verschiedene Nachrichten reagieren muß. Dies können Tastenbetätigungen, Mausklicks oder auch Zeitereignisse usw. sein. Der Window-Manager leitet die entsprechenden Meldungen des GEMs an die Fenster, für welche die Meldungen bestimmt sind.

Um besser mit den Fenstern zu arbeiten, legt sich der Window-Manager eine Liste aller in diesem Prozeß kreierten bzw. geöffneten Fenster an. So weiß er immer, welches Fenster oben liegt bzw. in welcher Reihenfolge die Fenster auf dem Bildschirm erscheinen.

Das Konzept des Window-Managers wurde bereits in Kapitel 6.3 recht ausführlich erklärt. An dieser Stelle folgen nun alle Funktionsaufrufe an den Window-Manager.

a) Globale Variablen

GLOBAL WINDOWP sel_window;

„sel_window" ist ein Zeiger auf die Beschreibung eines Fensters. Es handelt sich um das Fenster, das zuletzt angeklickt wurde (selektiertes Fenster). Die Variable nimmt den Wert NULL an, wenn kein Fenster angeklickt wurde oder das Fenster nicht anklickbar ist, d.h. in diesem Fenster kein Objekt selektiert werden kann. Mit Hilfe dieser Variablen kann das Modul MENU entscheiden, auf welches Objekt z.B. die Aktion „Hilfe" oder „Info" angewandt werden soll.

Es kann immer nur maximal ein Fenster selektiert sein. Ähnlich wie im Desktop soll beim Anklicken eines Fensters, in welchem keine Objekte selektiert sind, automatisch das zuletzt selektierte Fenster deselektiert werden.

GLOBAL SET sel_objs;

Es handelt sich um die Menge von (maximal 1024) Objekten, die im Fenster „sel_window" angeklickt sind. Ist die Menge nicht so einfach abzählbar, so muß diese Variable nicht unbedingt benutzt werden. Für ein Fenster kann auch ein anderer Algorithmus benutzt werden, der sich angewählte Objekte merkt (z.B. die Verwaltung einer linearen Liste o.ä.).

b) Globale Funktionen

WINDOWP search_window (WORD class, WORD mode, WORD icon);

Es kann nach dem obersten Fenster gesucht werden, welches einer bestimmten Klasse angehört, einen bestimmten Modus hat und zu einem bestimmten Piktogramm gehört.

search_window: Zeiger auf das gefundene Fenster oder NULL, wenn kein Fenster der Spezifikation genügt

class : Klasse des Fensters oder NIL, wenn die Klasse keine Rolle spielt
mode : Modus, in dem sich das Fenster befindet:
SRCH_CLOSED: Es wird ein geschlossenes Fenster gesucht
SRCH_OPENED: Es wird ein geöffnetes Fenster gesucht
SRCH_ANY : Beide Möglichkeiten
icon : Piktogramm, das zu dem eben genannten Fenster gehört, oder NIL, wenn es an kein Piktogramm gekoppelt ist

Ein Beispiel wird unter anderem im Modul CLIPBRD gegeben. Beim Öffnen des Klemmbrettes wird zunächst nach einem bereits offenen Klemmbrettfenster gesucht (CLASS_CLIPBRD, SRCH_OPENED, icon). Ist es vorhanden, so wird es lediglich nach oben gebracht.

Im anderen Fall wird nach einem geschlossenen Klemmbrettfenster gesucht (CLASS_CLIPBRD, SRCH_CLOSED, icon). Ist keines vorhanden, so wird eines kreiert. Dann wird das kreierte oder das bereits gefundene geschlossene Fenster geöffnet.

WINDOWP find_window (WORD wh);

Es kann nach einem Fenster gesucht werden, von dem das GEM-Window-Handle bekannt ist.

find_window: Zeiger auf das gefundene Fenster oder NULL, wenn kein Fenster mit einem bestimmten Handle existiert
wh : GEM-Window-Handle des zu suchenden Fensters

Die Funktion kann überall dort eingesetzt werden, wo man eine Fensterbeschreibung benötigt, aber von GEM nur ein Handle bekommt. Dies ist zum Beispiel nach jedem GEM-Fenster-Ereignis der Fall, aber auch beim Klick-Ereignis, wo man zunächst das Handle des angeklickten Fensters bestimmt, dann dessen Beschreibung (siehe Modul EVENT).

WINDOWP find_top (VOID);

Die Funktion liefert einen Zeiger auf die Fensterbeschreibung des obersten Fensters.

find_top: Zeiger auf das oberste Fenster oder NULL, wenn kein Fenster offen ist

Im Modul MENU wird die Funktion eingesetzt, um das oberste Fenster festzustellen. Abhängig davon können dann bestimmte Menüpunkte abgeschaltet werden (z.B. CUT/COPY/PASTE), wenn diese nicht für das oberste Fenster definiert sind.

BOOLEAN is_top (WINDOWP window);

Es kann getestet werden, ob ein bestimmtes Fenster ganz oben in diesem Prozeß liegt.

is_top : TRUE, wenn das zu testende Fenster oben liegt
window: Zeiger auf das Fenster, welches getestet werden soll

Es bietet sich an, die Funktion zu benutzen, um festzustellen, ob bestimmte Aktionen an einem Fenster durchgeführt werden dürfen. Beispielsweise sollten Fenster, in die Eingaben gemacht werden können, nur auf Eingaben reagieren, wenn sie ganz oben liegen. Sonst würden z.B. beim Schieben (nach rechts) von Fensterinhalten eventuell verdeckte Teile mitverschoben.

BOOLEAN any_open (BOOLEAN incl_desk, BOOLEAN incl_closer);

Es kann getestet werden, ob irgendwelche Fenster offen sind. Dabei kann noch spezifiziert werden, ob der Desktop auch zu den offenen Fenstern zählt (er kann meist nicht geschlossen werden), oder ob das Vorhandensein der Schließbox berücksichtigt werden soll.

any_open : TRUE, wenn mindestens ein Fenster dieses Prozesses offen ist, auf das die Spezifikation zutrifft, FALSE sonst
incl_desk : TRUE, wenn der Desktop ebenfalls berücksichtigt werden soll, FALSE sonst
incl_closer: TRUE, wenn nur Fenster berücksichtigt werden sollen, die auch eine Schließbox haben, FALSE sonst

Von dieser Funktion kann z.B. das Ein/Ausschalten eines Menüs, welches „Schließen" heißt, abhängig gemacht werden (siehe auch Modul MENU).

WORD num_windows (WORD class, WORD mode, WINDOWP winds []);

Die Anzahl der Fenster einer oder aller Klassen kann ermittelt werden. Dabei kann noch angegeben werden, ob offene oder geschlossene Fenster berücksichtigt werden sollen. Ein zusätzlicher Parameter gibt alle Fensterzeiger zurück, so daß mit allen gefundenen Fenstern Operationen durchgeführt werden können.

num_windows: Anzahl der Fenster, auf die die Spezifikationen zutreffen
class : Klasse der Fenster oder NIL, wenn die Klasse keine Rolle spielt
mode : Modus, in dem sich die Fenster befinden:
SRCH_CLOSED: Es werden geschlossene Fenster gesucht
SRCH_OPENED: Es werden geöffnete Fenster gesucht
SRCH_ANY : Beide eben genannten Möglichkeiten
winds : Zeiger auf ein Feld von Fensterzeigern, die auf die gefundenen Fenster deuten, oder NULL, wenn man die Fenster nicht weiterverarbeiten möchte

Beispiele finden sich in fast allen Modulen. Es wird jeweils die Anzahl der Fenster einer Klasse ermittelt, damit zwei Fenster einer Klasse, die hintereinander geöffnet werden, nicht direkt übereinander zu liegen kommen.

Um die Anzahl aller Fenster der Klasse CLASS_META zu ermitteln, die offen sind, muß z.B.

```
num = num_windows (CLASS_META, SRCH_OPENED, NULL);
```

aufgerufen werden. Soll das Innere aller dieser offenen Fenster neu gezeichnet werden, so müssen die Fenster in einem Feld abgespeichert werden.

```
num = num_windows (CLASS_META, SRCH_OPENED, winds);
for (i = 0; i < num; i++) redraw_window (winds [i], &desk);
```

Dabei muß „winds" ein Feld sein, daß Platz für maximal 8 Fenster hat (maximale Anzahl offener Fenster in GEM), also z.B.

```
WINDOWP winds [MAX_GEMWIND];
```

```
WINDOWP create_window (UWORD kind, WORD class);
```

Eine neue Inkarnation einer Klasse wird in Form eines Fensters kreiert. Die Art und Klasse des Fensters wird hier bereits festgelegt. Alle anderen für das Fenster typischen Variablen und Aktionen werden nach dem erfolgreichen „create_window" initialisiert. Nach dem Kreieren wird das Fenster nicht geöffnet.

create_window: Zeiger auf das erzeugte Fenster oder NULL, wenn kein Speicherplatz mehr für das Fenster zur Verfügung steht
kind : Art der Komponenten eines Fensters (siehe AES.H)
class : Klasse des Fensters

Die Anzahl der maximal zu kreierenden Fenster hängt vom zur Verfügung gestellten Speicherplatz ab (siehe „init_windows" weiter unten). Ist ein Fenster kreiert, so wird es vom Window-Manager in eine Liste aufgenommen, in der alle offenen und geschlossenen Fenster eingetragen sind. Da das Fenster noch nicht geöffnet ist, steht es ganz unten in der Liste. Das oberste geöffnete Fenster steht in der Liste an der Stelle mit Index 0.

Das Kreieren eines Fensters benötigt noch kein GEM-Window-Handle. Der GEM-Aufruf „wind_create" erfolgt erst beim Öffnen des Fensters. Dadurch können auch mehr als acht Fenster erzeugt werden. Es können lediglich nur maximal acht Fenster offen sein.

```
VOID delete_window (WINDOWP window);
```

Ein Fenster wird gelöscht und damit aus der Liste des Window-Managers entfernt. Der freigewordene Speicherplatz wird zum Kreieren neuer Fenster benutzt.

window: Zeiger auf das Fenster, welches gelöscht werden soll

Bevor das Fenster gelöscht wird und damit eventuell Datenstrukturen freigegeben werden, die den Fensterinhalt anzeigen, geschieht eine Testabfrage. Dazu muß eine Funktion in der Komponente „test" der Fensterstruktur eingeklinkt sein. Diese Funktion wird dann mit dem Parameter „window" und „DO DELETE" aufgerufen. Liefert sie TRUE, so wird das Fenster endgültig gelöscht. An diese Funktion kann man also z.B eine Alertbox hängen, die abfragt, ob das Fenster wirklich gelöscht werden soll.

Ist das Fenster noch offen, wenn es gelöscht werden soll, so wird es zunächst geschlossen. Vor dem Freigeben der Datenstruktur wird die Funktion „delete" aufgerufen, die in der Fensterstruktur eingeklinkt werden kann. In dieser Funktion können noch einmal Daten gerettet werden.

BOOLEAN open_window (WINDOWP window);

Ein vorher kreiertes Fenster wird geöffnet. Auch ein bereits offenes Fenster kann noch einmal geöffnet werden (mehrstufiges Öffnen). Es muß dann auch genauso oft geschlossen werden, wie es geöffnet wurde. Ähnliches gilt für den Desktop, bei dem ein Klicken auf die Schließbox nicht unbedingt bedeutet, daß das Fenster tatsächlich geschlossen wird. Vielmehr wird dort um eine Ebene zurückgegangen.

open_window: TRUE; wenn das Öffnen des Fensters geklappt hat, FALSE sonst
window : Zeiger auf das Fenster, welches geöffnet werden soll

War das Fenster noch nicht offen, so wird mittels „wind_create" ein neues Fenster angelegt. Dies gilt aber nur für den Fall, daß es sich um ein richtiges Fenster handelt. Ein eigener Desktop wird insofern berücksichtigt, als kein „wind_create" aufgerufen wird. Außerdem wird das Handle auf 0 gesetzt.

Ist bereits ein anderes Fenster des gleichen Prozesses offen, so wird für dieses die Funktion „untop" der Fensterstruktur aufgerufen, da es nach unten kommt. Dann wird für das zu öffnende Fenster die Funktion „open" der Fensterstruktur aufgerufen. Ist das Fenster nun der Desktop, wird es automatisch beim AES angemeldet, ansonsten wird das Fenster normal geöffnet, wobei vorher Name, Infozeile und Schieber gesetzt werden. Anschließend wird es noch in die Liste des Window-Managers eingetragen.

VOID close_window (WINDOWP window);

Ein zuvor geöffnetes Fenster wird geschlossen. Auf ein bereits geschlossenes Fenster hat die Funktion keine Wirkung.

window: Zeiger auf das Fenster, welches geschlossen werden soll

Vor dem Schließen wird zunächst die Funktion „test" mit den Parametern „window" und „DO_CLOSE" der Fensterstruktur aufgerufen. Existiert keine solche Funktion oder liefert diese den Wert TRUE zurück, so wird mit dem Schließen des Fensters fortgefahren. An diese Funktion kann z.B. eine Alertbox gehängt werden, die abfragt, ob das Fenster auch wirklich geschlossen werden soll.

War das Fenster selektiert, so wird es zunächst deselektiert. Ist das Fenster der Desktop, so wird es beim AES abgemeldet. Damit erscheint wieder der vor dem Programmstart bekannte Desktop. Ansonsten wird das Fenster geschlossen. Wurde das Fenster mehrstufig geöffnet, so verschwindet es aber erst dann vom Bildschirm, wenn es zum letzten Mal geschlossen wird. Für die alte GEM-Version wird noch das Schließen für Accessory-Fenster berücksichtigt. Falls das Accessory die Meldung AC_CLOSE erhält, dürfen die Fenster weder geschlossen noch gelöscht werden, da sonst der Rechner gnadenlos abstürzt oder hängenbleibt.

Nun wird die Funktion „close" der Fensterstruktur aufgerufen. An dieser Stelle können bestimmte Aktionen getätigt werden, wie z.B. das Zeichnen einer schrumpfenden Box oder das Abspeichern eines Dokumentes.

Wird das Fenster zum letzten Mal geschlossen, so wird es gelöscht, wenn das Flag WI_RESIDENT nicht gesetzt war. Es laufen dann die gleichen Aktionen wie bei „delete_window" ab.

Falls nun ein anderes Fenster dieses Prozesses nach oben kommt, wird für dieses die Funktion „top" der Fensterstruktur aufgerufen.

VOID close_top (VOID);

Das oberste Fenster wird geschlossen. Dabei spielt es keine Rolle, ob dieses Fenster eine Schließbox hat oder nicht. Wurde das Fenster mehrstufig geöffnet, so wird um eine Stufe zurückgegangen.

VOID close_all (BOOLEAN delete, BOOLEAN close_desk);

Es werden alle Fenster geschlossen und/oder gelöscht. Dabei kann angegeben werden, ob der Desktop ebenfalls geschlossen werden soll oder nicht.

delete : TRUE, wenn die Fenster nach dem Schließen auch gelöscht werden sollen, FALSE sonst
close_desk : TRUE, wenn der Desktop mit geschlossen werden soll, FALSE sonst

VOID draw_window (WINDOWP window);

Der Fensterinhalt wird gezeichnet. Soll nur ein Teil gezeichnet werden, so muß vorher das Clipping gesetzt werden. Die Rechteckliste wird hier nicht berücksichtigt.

window: Zeiger auf das Fenster, welches gezeichnet werden soll

Da der Window-Manager nicht weiß, was sich genau im Fenster befindet, wird die Routine „draw" der Fensterstruktur aufgerufen. In dieser Routine entscheidet die Applikation, was gezeichnet werden soll. Ist ein Objektbaum ins Fenster eingeklinkt, so wird dieser automatisch gezeichnet.

VOID redraw_window (WINDOWP window, CONST RECT *area);

Ein bestimmter Bereich eines Fensters wird unter Berücksichtigung der Rechteckliste neu gezeichnet. Die Maus wird automatisch versteckt.

window: Zeiger auf das Fenster, welches gezeichnet werden soll
area : Zeiger auf den Bereich, welcher gezeichnet werden soll

Das Clipping des Bereichs wird gesetzt, bevor das Fenster gezeichnet wird. Hat das Fenster eine Menüzeile, so wird diese automatisch gezeichnet.

VOID top_window (WINDOWP window);

Ein Fenster wird nach oben gebracht.

window: Zeiger auf das Fenster, welches nach oben gebracht werden soll

Ist das Flag „WI_NOTOP" in der Fensterstruktur gesetzt, so wird das Fenster nicht nach oben gebracht. Für das alte GEM wird zusätzlich noch die Meldung „WM_UNTOPPED" dermaßen simuliert, daß das zweitoberste Fenster vorher „nach unten gebracht wird", d.h. von diesem Fenster wird die Funktion „untop" der Fensterstruktur angesprungen. Bei neueren GEM-Versionen (ab 2.X), wird vom GEM die Meldung „WM_UNTOPPED" gebracht, so daß die entsprechende Routine vom Event-Manager aufgerufen wird.

Für das Fenster, welches nach oben gebracht werden soll, wird die Funktion „top" der Fensterstruktur aufgerufen. An dieser Stelle können Aktionen ablaufen wie das Restaurieren von Zuständen, die geherrscht haben, bevor das Fenster nach unten gekommen ist.

VOID untop_window (WINDOWP window);

Ein Fenster, welches nach unten gebracht wird, kann eine Aktion durchführen.

window: Zeiger auf das Fenster, welches nach unten gebracht wird

Die Funktion wird normalerweise nur vom Event-Manager (siehe Modul EVENT) aufgerufen, wenn dieses die Meldung „WM_UNTOPPED" bekommen hat. Dann kann das *Fenster, welches nach unten kommt, noch eine Aktion durchführen (z.B. Retten des Fensterinhaltes, etc.).*

VOID scroll_window (WINDOWP window, WORD dir, LONG delta);

Der Inhalt eines Fensters wird um eine beliebige Anzahl von Pixeln in eine von vier Richtungen gescrollt. Die Maus wird automatisch versteckt.

window: Zeiger auf das Fenster, welches gescrollt werden soll
dir : Richtung, in die gescrollt werden soll, nämlich
HORIZONTAL: Links-Rechts-Scrolling
VERTICAL : Aufwärts-Abwärts-Scrolling
delta : Anzahl der Pixel, um die gescrollt werden soll, wobei gilt:
Richtung HORIZONTAL und delta > 0 ":" Links-Scrolling
Richtung HORIZONTAL und delta > 0 ":" Rechts-Scrolling
Richtung VERTICAL und delta > 0 = ":" Aufwärts-Scrolling
Richtung VERTICAL und delta < 0 ":" Abwärts-Scrolling

Die Funktion berücksichtigt verschiedene Sonderfälle. Wird ein Scrolling mit einem Wert „delta" aufgerufen, der größer ist als der scrollbare Fensterbereich, so wird nicht gescrollt, sondern der Scrollbereich des Fensters gleich neu gezeichnet. Dies kommt immer dann vor, wenn der Benutzer den Schieber um mehr als eine Seite bewegt.

Im anderen Fall wird exakt berechnet, wie viele Pixel gescrollt werden müssen. Dies gilt vor allem dann, wenn das Fenster nicht mehr ganz im Desktop liegt, da hier Informationen, die sonst durch Scrolling verfügbar sind, durch den Desktop „weggeclippt" sind. Die fehlende Information wird später durch ein entsprechendes Redraw verlangt. Viele Programme (z.B. die Editoren von Turbo C oder Laser C) sind offensichtlich außerstande, den Fensterinhalt zu scrollen, wenn dieser nur teilweise im Desktop liegt. Sie restaurieren dann jeweils den gesamten Fensterinhalt, was jedoch sehr langsam ist.

Das Scrolling selbst geschieht unter Zuhilfenahme der Rechteckliste. Dann geschieht, wenn das Fenster ganz sichtbar ist, ein echtes Scrolling und zwar unabhängig davon, wo das Fenster liegt. Das bedeutet, daß auch im Hintergrund gescrollt werden kann. Wird das Fenster von vielen anderen Fenstern verdeckt, ist dieser Algorithmus nicht mehr anwendbar, da Information, die durch Scrolling verfügbar sein müßte, nicht mehr existiert (z.B. Aufwärts-Scrolling, wenn die unteren oder oberen Zeilen eines Fensters teilweise durch ein anderes verdeckt sind). In diesem Fall geschieht das Scrolling durch permanentes Zeichnen des Fensterinhaltes. Das ist zwar nicht so schnell, aber es funktioniert.

VOID arrow_window (WINDOWP window, WORD arrow, WORD amount);

Ein Fenster reagiert auf das Anklicken eines Pfeils.

window: Zeiger auf das Fenster, dessen Pfeile angeklickt wurden
arrow : Pfeil, der angeklickt wurde, wobei gilt (siehe AES.H):
WA_UPPAGE: Seite nach oben
WA_DNPAGE: Seite nach unten
WA_UPLINE: Zeile nach oben
WA_DNLINE: Zeile nach unten
WA_LFPAGE: Seite nach links
WA_RTPAGE: Seite nach rechts
WA_LFLINE: Zeile nach links
WA_RTLINE: Zeile nach rechts
amount : Anzahl der Einheiten, um die gescrollt werden soll

Nachdem der Benutzer einen Pfeil angeklickt hat, sollte diese Routine vom Event-Manager (Modul EVENT) aufgerufen werden. Je nach Wert des Parameters „arrow" wird die neue Position innerhalb des Dokumentes berechnet. Falls die neue Position über den Rand hinausgeht, wird dies korrigiert. So führt das Anklicken des Aufwärts-Pfeils nicht zu einem Scrolling oder Neuzeichnen des Fensterinhalts, wenn bereits der obere Rand erreicht ist. Einige Programme tun dies aber trotzdem.

Nachdem die neue Position berechnet wurde, wird die Funktion „arrow" der Fensterstruktur mit den Parametern „window", „dir", „oldpos" und „newpos" aufgerufen. Letztere geben die alte und neue Position in den Einheiten an, die durch die Variablen „window->xfac" bzw. „window->yfac" festgelegt wurden. In dieser Funktion des Fensters kann nun entsprechend reagiert werden. Beispielsweise können dort neue Zeilen für das Redraw zur Verfügung gestellt werden oder neue Datensätze einer Datenbankliste geholt werden u.s.w. Beispiele sind in allen Modulen angegeben, in denen Fenster gescrollt werden können.

Ist ein Objektbaum in das Fenster eingeklinkt, so geschieht das Reagieren auf die Pfeile automatisch.

VOID h_slider (WINDOWP window, WORD new_value);

Ein Fenster reagiert auf das Bewegen des horizontalen Schiebers.

window : Zeiger auf das Fenster, dessen Schieber bewegt wurde
new_value: Vom GEM gelieferter neuer Wert für den Schieber (0 – 1000)

Nachdem der neue Schieberwert berechnet wurde, wird wie beim Anklicken eines horizontalen Pfeils reagiert.

VOID v_slider (WINDOWP window, WORD new_value);

Ein Fenster reagiert auf das Bewegen des vertikalen Schiebers.

window : Zeiger auf das Fenster, dessen Schieber bewegt wurde
new_value : Vom GEM gelieferter neuer Wert für den Schieber (0 – 1000)

Nachdem der neue Schieberwert berechnet wurde, wird wie beim Anklicken eines vertikalen Pfeils reagiert.

VOID set_sliders (WINDOWP window, WORD which, WORD mode);

Die Schieberpositionen und/oder die Schiebergrößen werden gesetzt.

window: Zeiger auf das Fenster, dessen Schieber gesetzt werden sollen
which : Angabe der Schieber, wobei gilt
HORIZONTAL: Horizontaler Schieber
VERTICAL : Vertikaler Schieber
mode : Operation, die durchgeführt werden soll, wobei gilt
SLPOS : Position des Schiebers neu setzen
SLSIZE: Größe des Schiebers neu setzen

Beim Aufruf können alle Kombinationen benutzt werden, so daß Position und Größe von beiden Schiebern in einem Durchgang verändert werden können. Der Aufruf würde dann lauten

set_sliders (window, HORIZONTAL + VERTICAL, SLPOS + SLSIZE);

Das Zeichnen der Schieber wird optimiert, so daß die Schieber nur dann neu gezeichnet werden, wenn sich ihr Wert tatsächlich geändert hat. Zwei aufeinanderfolgende Aufrufe von „set_sliders" ergibt somit kein unruhiges Zucken der Schieber auf dem Bildschirm. Dies ist sonst häufig der Fall, wenn die Auflösung der Schieber (1000 Punkte) nicht ausreicht, also z.B. mehr als 1000 Zeilen oder Spalten in einem Fenster stehen.

VOID snap_window (WINDOWP window, RECT *new, WORD mode);

Ein Fenster wird auf bestimmte Positionen des Bildschirms eingerastet. Dies ist für Zeichenoperationen sinnvoll. Zeichenketten werden auf Bytegrenzen schneller gezeichnet, da der Rechner dann keine Bit-Schiebe-Operationen ausführen muß. Außerdem müssen sich Menüzeilen immer auf geraden Y-Positionen befinden, da sonst das Muster, welches Menüeinträge grau darstellt, bestimmte Buchstaben entstellen würde. Darüberhinaus sollten sich bestimmte Muster (z.B. das „Grau" des Desktops) ebenfalls immer auf geraden X- und Y-Positionen befinden.

window: Zeiger auf das Fenster, welches eingerastet werden soll
new : Zeiger auf ein Rechteck, welches zunächst die aktuelle und nach dem Verändern die neue Position angibt

mode : Art des Einrastens, entweder nach dem Verschieben oder nach dem Vergrößern/Verkleinern des Fensters, wobei gilt
MOVED: Das Fenster wurde bewegt
SIZED : Das Fenster wurde vergrößert oder verkleinert

Die Funktion wird nach dem Verschieben oder Vergrößern/Verkleinern vom Window-Manager aufgerufen. Zunächst wird abgeprüft, ob der linke Rand des Fensteräußeren genau am linken Bildschirmrand liegt. Ist dies der Fall, so wird das Fenster um eine weitere Position ins Negative verschoben. Da der Rand des Fensters genau ein Pixel breit ist, kommt das Fensterinnere dabei genau an der X-Position null zu liegen. Damit kann das Fensterinnere leicht auf Bytegrenzen einrasten, ohne daß sieben Bits verlorengehen.

Verschiedene Programme wie z.B. Tempus gehen auf ähnliche Art vor. Bei anderen Programmen wie z.B. Turbo-C ist es nicht möglich, das Fenster an den linken Rand zu schieben, da dieses nicht um eine weitere Position nach links gebracht werden kann, so daß das Fensterinnere nicht auf einer Bytegrenze zu liegen kommt. Aus diesem Grund kann man das Fenster nicht ganz an den linken Rand schieben, es gehen immer sieben Pixel an Verschiebemöglichkeiten verloren.

Die Funktion verhindert außerdem, daß ein Fenster zu klein werden kann. Während Fenster mit Schiebern nicht unter eine bestimmte Minimalgröße gebracht werden können, ist dies bei Fenstern ohne Schieber nicht der Fall. Unterschreitet die neue Größe bestimmte Grenzen (MIN WIDTH und MIN_HEIGHT), so werden diese korrigiert.

Danach wird die Funktion „snap" der Fensterstruktur aufgerufen. Dort kann jedes Fenster für sich entscheiden, welche Aktionen beim Einrasten durchgeführt werden sollen. So kann z.B. beim Vergrößern eines Fensters unter den unteren Rand eines Dokumentes die Y-Position innerhalb des Dokuments (doc.y) verkleinert werden, so daß das Fenster wieder den unteren Rand des Dokumentes, jedoch nun mehr Zeilen als vorher anzeigt.

Bei den Einrastaktionen der Fenster werden jeweils nur Differenzen berechnet, keine absoluten Positionen. So kann das Fenster leichter entscheiden, wieviele Zeilen oder Spalten nun dazu- oder weggekommen sind. Da jeweils immer nur Differenzen benutzt werden, muß sich das Fenster beim erstmaligen Öffnen auf einer der Positionen befinden, auf die jeweils eingerastet werden soll.

VOID full_ window (WINDOWP window);

Das Fenster reagiert auf das Anklicken der Full-Box in der rechten oberen Ecke.

window: Zeiger auf das Fenster, dessen Full-Box angeklickt wurde

Der Window-Manager sorgt von selbst dafür, daß das Fenster nur dann auf volle Größe gebracht wird, wenn es vorher geöffnet, bewegt, vergrößert oder verkleinert wurde. Das Zeichnen der Grow- und Shrinkbox wird über die globalen Funktionen „growbox" und „shrinkbox" aufgerufen, so daß mittels der globalen Variablen „grow_shrink" eingestellt werden kann, ob die Boxen gezeichnet werden oder nicht.

VOID size_window (WINDOWP window, CONST RECT *new);

Das Fenster reagiert auf ein Vergrößern oder Verkleinern.

window: Zeiger auf das Fenster, welches vergrößert oder verkleinert wird
new : Zeiger auf das Rechteck, welches die gewünschte neue Größe des Fensters angibt

Bevor die neue Größe des Fensters gesetzt wird, wird dem Fenster die Möglichkeit gegeben, via „snap_window" zu bestimmen, ob das Fenster auf bestimmte Positionen eingerastet werden soll. Die Schieber des Fensters werden automatisch gesetzt.

VOID move_window (WINDOWP window, CONST RECT *new);

Das Fenster reagiert auf ein Bewegen.

window: Zeiger auf das Fenster, welches bewegt wird
new : Zeiger auf das Rechteck, welches die gewüschte neue Position des Fensters angibt

Bevor die neue Position des Fensters gesetzt wird, wird dem Fenster die Möglichkeit gegeben, via „snap_window" zu bestimmen, ob das Fenster auf bestimmte Positionen eingerastet werden soll. Die Schieber des Fensters werden automatisch gesetzt.

WORD drag_to_window (WORD mox, WORD moy,
WINDOWP src window, WORD src_obj,
WINDOWP *dest_window, WORD *dest_obj);

Das Ziehen von Objekten aus einem Fenster in ein anderes wird unterstützt.

drag_to_window : Ergebnis der Ziehoperation, wobei folgende Konstanten vordefiniert sind:
DRAG_OK : Die Drag-Operation ist ok.
DRAG_SWIND : Das Objekt wurde innerhalb des gleichen Fensters verschoben.
DRAG_SCLASS : Das Objekt wurde auf ein Fenster der gleichen Klasse gezogen, also z.B. von einem Dateifenster in ein anderes, aber nicht von einem Dateifenster in ein Papierkorbfenster.
DRAG_NOWIND : Das Zielfenster, in welches das Objekt gezogen wurde, gehört nicht zum gleichen Prozeß, kann also z.B. ein Fenster von einem Deskaccessory sein. In einer Multitasking-Umgebung sollte dem fremden Prozeß mittels „wind_apfind" eine Nachricht gesendet werden.
DRAG_NORCVR : Das Zielfenster gehört zwar zum eigenen Prozeß, stellt aber keine Routine zur Verfügung, um Drag-Operationen auszuwerten.

DRAG_NOACTN : Das Zielfenster besitzt zwar eine Routine, welche auf eine Drag-Operation reagiert, kann aber mit dem Objekt nichts anfangen.
mox : X-Koordinate der Maus beim Beenden der Zieh-Operation
moy : Y-Koordinate der Maus beim Beenden der Zieh-Operation
src_window : Zeiger auf das Quellfenster der Zieh-Operation
src_obj : Nummer des Objekts, welches gezogen wird
dest_window : Zeiger auf den Zeiger auf das Zielfenster oder NULL, wenn kein Zielfenster dieses Prozesses existiert
dest_obj : Zeiger auf das Zielobjekt oder NIL, wenn kein Zielobjekt existiert

Bei einer Zieh-Operation ruft das Fenster, in welchem Objekte angeklickt und verschoben wurden, die oben beschriebene Funktion auf. Dabei werden die aktuellen Maus-Koordinaten sowie das Quellfenster übergeben. Falls die Objekte nicht der gleichen Klasse angehören, sollte für jedes Objekt (src_obj) die Operation jeweils aufgerufen werden. Dies liegt daran, daß beim Verschieben von Objekten unterschiedlicher Art eventuell verschiedene Operationen ausgeführt werden müssen. So hat das gleichzeitige Verschieben des Klemmbretts und des Diskettensymbols in das Papierkorbfenster jeweils unterschiedliche Aktionen zur Folge.

Der Algorithmus läuft nun folgendermaßen ab: Zunächst wird das Zielfenster ausfindig gemacht. Gehört dieses nicht zum eigenen Prozeß, so wird der Wert „DRAG_NOWIND" zurückgegeben.

Danach wird das Zielobjekt ausfindig gemacht, falls innerhalb des Fensters ein Objektbaum eingeklinkt ist. Ansonsten erhält „dest_obj" den Wert NIL.

Ist nun in das Zielfenster eine Funktion „drag" in die Fensterstruktur eingeklinkt, so wird diese aufgerufen. Sie erhält als Parameter das Quellfenster, das Quellobjekt, das Zielfenster und das Zielobjekt. Damit kann das Zielfenster auf die Zieh-Operation reagieren. Dessen Rückgabewert wird wiederum an das Quellfenster geliefert, das nun ebenfalls reagieren kann. Das Zielfenster sollte je nach Objekttyp entsprechende Werte zurückliefern. Es kann dabei die oben beschriebenen Standardwerte benutzen oder eigene Werte, die größer als null sein sollten. Durch Benutzung der gleichen Werte können sich Quell- und Zielfenster somit gegenseitig absprechen, so daß nur das Quell-, nur das Ziel- oder beide Fenster reagieren.

Ist keine Funktion „drag" eingeklinkt, so ist der Empfänger wohl desinteressiert, weshalb der Wert „DRAG_NORCVR" zurückgeliefert wird.

Beispiele der Anwendung dieser Funktion sind in den Modulen CLIPBRD und DESKTOP gegeben.

VOID click_window (WINDOWP window, MKINFO *mk);

Ein Fenster wurde vom Benutzer angeklickt (selektiert).

window: Zeiger auf das Fenster, welches angeklickt wurde
mk : Zeiger auf die Maus- und Tastaturinformation

Klickt der Benutzer auf ein Fenster, so wird vom Modul EVENT diese Funktion aufgerufen. Falls eine Funktion „click" in die Fensterstruktur eingeklinkt wurde, so wird diese mit den Parametern „window" und „mk" aufgerufen. Das Fenster kann nun Objekte selektiert, also z.B. invers darstellen. Das Deselektieren eines anderen angewählten Fensters muß in dieser Funktion stattfinden. Würde dies automatisch vorher geschehen, so könnte z.B. der Desktop beim Anklicken einer Funktionstaste nicht mehr feststellen, ob ein Objekt in einem anderen Fenster bereits angeklickt wurde, auf das die Funktionstaste reagieren muß.

Existiert keine Funktion „click", so wird automatisch das zuletzt selektierte Fenster deselektiert, falls es nicht das angeklickte Fenster war.

VOID unclick_window (WINDOWP window);

Ein Fenster wird deselektiert. Diese Funktion ist das Gegenstück zu „click_window".

window: Zeiger auf das Fenster, welches deselektiert werden soll

Die Funktion „unclick" der Fensterstruktur wird aufgerufen, falls sie vorhanden ist. In dieser müssen die Objekte, die in diesem Fenster selektiert sind, deselektiert werden. Danach werden die globalen Variablen „sel_window" und „sel_objs" zurückgesetzt.

BOOLEAN key_window (WINDOWP window, MKINFO *mk);

Der Benutzer hat eine Taste gedrückt, welche an ein Fenster weitergeleitet wird.

key_window: TRUE, wenn das Fenster den Tastendruck verarbeitet hat, FALSE sonst
window : Zeiger auf das Fenster, für welches der Tastendruck bestimmt ist
mk : Zeiger auf die Maus- und Tastaturinformation

Der Tastendruck wird nur an das Fenster geleitet, wenn es offen ist und zusätzlich die Funktion „key" der Fensterstruktur gesetzt ist. In diesem Fall wird der Funktionswert von „key" zurückgeliefert, FALSE sonst. Damit kann das Fenster entscheiden, ob es auf einen Tastendruck reagieren soll (TRUE), oder ihn ignorieren soll (FALSE). In letzterem Fall wird der Event-Manager die Information an das nächste Fenster weiterleiten, welches sich ebenfalls entscheiden kann, den Tastendruck zu verarbeiten oder zu ignorieren (siehe „key_all").

BOOLEAN key_all (MKINFO *mk);

Eine Tastenkombination wird allen Fenstern zur Verarbeitung angeboten.

key_all: TRUE, wenn mindestens ein Fenster auf eine Taste reagiert hat, FALSE sonst
mk : Zeiger auf die Maus- und Tastaturinformation

Der Event-Manager ruft „key_all" auf, wann immer der Benutzer eine Taste gedrückt hat. In dieser Funktion werden nun alle Fenster abgearbeitet. Zuerst wird dem obersten Fenster die Taste angeboten. Kann dieses Fenster die Taste verarbeiten, wird es den Funktionswert TRUE zurückliefern, und die Verarbeitung wird abgebrochen. Ansonsten wird dem nächsten Fenster (von oben) die Taste angeboten. Dies wiederholt sich, bis alle Fenster abgearbeitet wurden. Das letzte Fenster wird meist der Desktop sein.

Beispiele werden in allen Fenstern gegeben, die Tasten verarbeiten. Wichtig für das Verarbeiten von Tasten ist es, die Taste vor allem dann an das nächste untere Fenster weiterzuleiten, wenn es sich um eine Funktionstaste, Control- oder Alternate-Kombination handelt, auf die noch nicht reagiert wurde. Dann kann am Ende der Desktop auf die Taste reagieren und entsprechende Menüs können ausgewählt werden.

VOID timer_window (WINDOWP window);

Der Zeitvorgabe für ein Fenster ist abgelaufen. Dieses kann dann darauf reagieren.

window: Zeiger auf das Fenster, welches auf einen Zeitablauf reagieren soll

Die Funktion wird von „timer_all" angesprungen, damit die Fenster auf den Zeitablauf reagieren können. Dies geschieht aber nur, wenn eine Funktion in die Komponente „timer" der Fensterstruktur eingeklinkt ist. Man kann sie aber auch selbst aufrufen, um den Zeitablauf vorzeitig zu forcieren (z.B. durch einen Tastendruck oder zu Testzwecken).

VOID timer_all (LONG milli);

Allen Fenstern wird eine Zeitscheibe zur Verfügung gestellt, in denen sie Funktionen ausführen können. Damit ist eine Art Multitasking innerhalb eines Programmes möglich.

milli: Anzahl der Millisekunden, die seit dem letzten Zeitereignis vergangen sind

Die Funktion wird vom Event-Manager aufgerufen, wenn dieser auf das Eintreten eines Zeitereignisses gewartet hat. Die Anzahl der dabei verstrichenen Millisekunden wird an diese Funktion übergeben. Sie überprüft dann für alle Fenster (von oben angefangen), ob die Anzahl der Millisekunden, auf die das Fenster gewartet hat, größer als null ist. In diesem Fall wird „milli" zu den im Fenster verstrichenen Millisekunden (count) hinzuaddiert. Falls dieser Wert den im Fenster eingestellten Wert überschreitet, so wird der Zähler zurückgesetzt und das Zeitereignis für dieses Fenster ausgelöst.

Der Timer, der von der Applikation innerhalb von GEM benutzt wird, ist allerdings für kleine Werte nicht sehr genau, da die Auflösung zu gering ist. Aus diesem Grund bieten sich Werte um 100 Millisekunden an, die eine Applikation warten kann. Diese Werte stimmen recht genau mit der tatsächlich verstrichenen Zeit überein. Falls dieser Wert zu groß ist, das Fenster also öfter als jede zehntel Sekunde eine Aktion ausführen muß,

kann der Wert in „evnt_multi" auch verringert werden. Dann tritt das Zeitereignis öfter ein, jedoch stimmt dieses dann nicht mehr mit der tatsächlich verstrichenen Zeit überein.

VOID get_border (WINDOWP window, WORD obj, RECT *border);

Der Umriß eines Objekts innerhalb eines Fensters kann bestimmt werden. Dies funktioniert aber nur, wenn ein Objektbaum in die Fensterstruktur eingeklinkt wurde.

window: Zeiger auf das Fenster, von welchem ein Umriß bestimmt werden soll
obj : Index des Objekts
border : Zeiger auf das Rechteck, welches den Umriß enthält

Falls das Fenster nicht existiert, nicht offen ist, kein Objektbaum existiert oder das Objekt NIL ist, wird der Umriß auf null gesetzt, d.h. X- und Y-Koordinate sowie Breite und Höhe sind null. Wird dann ein „graf_growbox" oder ein „graf_shrinkbox" mit diesem Rechteck durchgeführt, so wird es automatisch von der Bildschirmmitte geöffnet. Dieser Effekt ist insofern günstig, als dann ein Objekt zusammenschrumpfen kann, auch wenn das Vaterobjekt oder Fenster nicht mehr existiert.

VOID draw_object (WINDOWP window, WORD obj);

Ein Objekt in einem Fenster wird unter Berücksichtigung der Rechteckliste gezeichnet. Dazu muß allerdings ein Objektbaum über die Komponente „object" in das Fenster eingeklinkt worden sein.

window: Zeiger auf das Fenster, in welchem ein Objekt gezeichnet werden soll
obj : Index des Objekts

```
VOID drag_boxes (WORD num_objs, CONST RECT *boxes,
                 WINDOWP inv_window, SET inv_objs, RECT *diff,
                 CONST RECT *bound);
```

Es können mehrere Rechtecke gleichzeitig verschoben werden. Außerdem kann ein Fenster angegeben werden, in welchem Objekte invertiert werden, die während des Zieh-Vorgangs überstrichen werden.

num_objs : Anzahl der Objekte, die verschoben werden
boxes : Zeiger auf eine Rechteckliste, die „num_objs" Rechtecke enthält, welche die zu verschiebenden Rechtecke beschreiben
inv_window : Zeiger auf ein Fenster, in welchem Objekte während des Ziehens invertiert werden
inv_objs : Objekte, welche innerhalb dieses Fensters selektiert werden dürfen

diff : Zeiger auf ein Rechteck, welches die Differenz der verschobenen Rechtecke zum Ausgangspunkt angibt
bound : Zeiger auf ein Rechteck, in welchem sich die zu verschiebenden Rechtecke bewegen dürfen, oder NULL, wenn sich diese innerhalb des gesamten Desktops bewegen dürfen

Die Funktion simuliert im Prinzip „graf_dragbox" des AES. Allerdings dürfen hier mehrere Rechtecke angegeben werden. Außerdem können noch Objekte eines Fensters invertiert werden. Das Zeichnen der Rechtecke ist eine nichttriviale Aufgabe, da diese auf jedem Hintergrund korrekt aussehen sollen. Dies gilt vor allem für das Muster des Desktop. Je nach Position des Rechtecks und dem Zeichenalgorithmus der vier Linien müssen andere Linienmuster benutzt werden. Dieser Algorithmus funktioniert zumindest für alle Rechner, bei denen der linke obere Eckpunkt (0, 0) beim Desktop-Muster nicht gesetzt ist. Falls ein Grafikprozessor seine Arbeit verrichtet, der das „Grau" des Desktops invertiert darstellt bzw. andere Algorithmen benutzt, um die Linien zu zeichnen, ist die Routine natürlich nicht mehr ganz korrekt. Ein Programm, das z.B. einen unkorrekten Algorithmus verwendet, ist das Resource-Construction-Set von Kuma. Bewegt man dort Piktogramme, so erscheinen je nach deren Position abwechselnd schwarze und weiße Umrisse.

Als Ergebnis wird in „diff.w" und „diff.h" die Differenz der Boxen nach dem Ziehvorgang zurückgegeben. Darüberhinaus ist in „diff.x" und „diff.y" die letzte Mausposition gespeichert. Beispiele für diese Funktion finden sich in den Modulen CLIPBRD und DESKTOP.

VOID scroll_area (CONST RECT *area, WORD dir, WORD delta);

Ein rechteckiger Bildschirmbereich wird um eine beliebige Anzahl von Pixeln in eine von vier Richtungen gescrollt. Die Maus wird automatisch versteckt.

area : Zeiger auf das Rechteck, welches gescrollt werden soll
dir : Richtung, in die gescrollt werden soll, nämlich
HORIZONTAL: Links-Rechts-Scrolling
VERTICAL : Aufwärts-Abwärts-Scrolling
delta : Anzahl der Pixel, um die gescrollt werden soll, wobei gilt:
Richtung HORIZONTAL und delta > 0 ":" Links-Scrolling
Richtung HORIZONTAL und delta < 0 ":" Rechts-Scrolling
Richtung VERTICAL und delta > 0 ":" Aufwärts-Scrolling
Richtung VERTICAL und delta < 0 ":" Abwärts-Scrolling

Im Gegensatz zu „scroll_window" kann hier auch ein kleinerer Bereich gescrollt werden, der nicht von der momentanen Fenstergröße abhängig ist. Dadurch kann z.B. eine einzelne Zeile eines Texteditorfensters um einen Buchstaben gescrollt werden, wenn in dieser Zeile ein Zeichen eingegeben wird, welches nicht das letzte Zeichen einer Zeile ist.

VOID clr_area (CONST RECT *area);

Ein rechteckiger Bildschirmbereich wird gelöscht (mit weiß gefüllt).

area: Zeiger auf den Bereich, der gelöscht werden soll

VOID clr_work (WINDOWP window);

Der Arbeitsbereich eines Fensters wird gelöscht (mit weiß gefüllt). Hat das Fenster eine Menüzeile, so wird diese nicht gelöscht.

window: Zeiger auf das Fenster, dessen Arbeitsbereich gelöscht werden soll

VOID clr_scroll (WINDOWP window);

Der Scrollbereich eines Fensters wird gelöscht (mit weiß gefüllt).

window: Zeiger auf das Fenster, dessen Scrollbereich gelöscht werden soll

VOID clr_left (WINDOWP window);

Der linke Rand eines Fensters wird gelöscht (mit weiß gefüllt).

window: Zeiger auf das Fenster, dessen linker Rand gelöscht werden soll

VOID clr_top (WINDOWP window);

Der obere Rand eines Fensters wird gelöscht (mit weiß gefüllt). Hat das Fenster eine Menüzeile, so wird diese nicht gelöscht.

window: Zeiger auf das Fenster, dessen oberer Rand gelöscht werden soll

VOID clr_right (WINDOWP window);

Der rechte Rand eines Fensters wird gelöscht (mit weiß gefüllt).

window: Zeiger auf das Fenster, dessen rechter Rand gelöscht werden soll

VOID clr_bottom (WINDOWP window);

Der untere Rand eines Fensters wird gelöscht (mit weiß gefüllt).
window: Zeiger auf das Fenster, dessen unterer Rand gelöscht werden soll

VOID set_redraw (WINDOWP window, CONST RECT *area);

Einem Fenster wird die Meldung gesendet, daß ein Teil seines Bereichs neu gezeichnet werden soll. Dies resultiert in der Nachricht „WM_REDRAW" an den Event-Manager.

window: Zeiger auf das Fenster, an das die Meldung gesendet werden soll
area : Zeiger auf den Bereich, der neu gezeichnet werden soll

VOID draw_mtitle (WINDOWP window, WORD title);

Der Menütitel eines offenen Fensters, welches eine Menüzeile enthält, wird gezeichnet. Dabei wird die Rechteckliste des Fensters berücksichtigt.

window: Zeiger auf das Fenster, dessen Menütitel gezeichnet werden soll
title : Index des Titels im Menübaum des Fensters

VOID draw_mbar (WINDOWP window);

Die Menüzeile eines Fensters, welches eine Menüzeile enthält, wird gezeichnet. Dabei wird die Rechteckliste des Fensters berücksichtigt.

window: Zeiger auf das Fenster, dessen Menüzeile gezeichnet werden soll

VOID menu_normal (WINDOWP window, WORD title, BOOLEAN normal);

Der Menütitel eines Fensters oder ein globaler Menütitel werden normal oder invers dargestellt. Dies entspricht dem Aufruf „menu_tnormal" der AES-Menü-Bibliothek, wobei hier noch ein Fenster übergeben werden kann.

window: Zeiger auf das Fenster, dessen Menütitel gezeichnet werden soll, oder NULL, wenn das globale Menü benutzt werden soll
title : Index des Titels im Menübaum des Fensters
normal : TRUE, wenn der Titel normal dargestellt werden soll, FALSE, wenn er invers dargestellt werden soll

Um zu gewährleisten, daß ein Programm auch als Accessory abläuft, sollte nicht „menu_tnormal" der AES-Bibliothek, sondern „menu_normal" des Window-Managers aufgerufen werden (siehe Modul MENU). Dann wird bei normalem Betrieb die globale Menüzeile benutzt, bei Betrieb als Accessory die Menüzeile im Desktop-Fenster.

```
BOOLEAN menu_manager (WINDOWP window, WORD mox, WORD moy,
                      WORD mobutton, WORD breturn);
```

Eine komplette Menü-Verwaltung innerhalb eines Fensters wird durchgeführt. Diese Routine wird vom Event-Manager (Modul EVENT) aufgerufen, wenn in ein Fenster geklickt wurde.

menu_manager: TRUE, wenn in die Menüzeile oder die Pfeile geklickt wurde, FALSE sonst
window : Zeiger auf das Fenster, in dessen Menüzeile geklickt wurde
mox : X-Position der Maus beim Anklicken
moy : Y-Position der Maus beim Anklicken
mobutton : Status der Knöpfe der Maus beim Anklicken
breturn : Anzahl der Klicks

Bei der Bedienung wird immer der linke Mausknopf benutzt. Zunächst wird getestet, ob sich die Mauskoordinaten überhaupt innerhalb der Menüzeile befinden. Ist dies der Fall, so wird geprüft, ob sich die Koordinaten innerhalb der Pfeile befinden. Bei positivem Test werden die Pfeile abgehandelt.

Ein Einfachklick in einen der Pfeile bewegt die Menüzeile in die entsprechende (umgekehrte) Richtung, ähnlich wie beim normalen Scrolling eines Fensters. Bei einem Doppelklick wird entweder auf das erste Menü (linker Pfeil) oder auf das letzte Menü (rechter Pfeil) positioniert. Diese Operationen wiederholen sich solange, bis die Pfeile verlassen werden.

Wurde nicht in einen der Pfeile geklickt, so wird zunächst die Funktion „updt_menu" der Fensterstruktur aufgerufen. Dort kann die Applikation noch schnell Menütitel oder Menüeinträge abschalten (grau darstellen).

Nun werden die Boxen, in denen sich die Menüeinträge befinden, an die aktuelle Bildschirmposition angepaßt. Es wird immer versucht, möglichst viele Menüeinträge zu zeigen. Für die horizontale Richtung gilt daher: Würde ein Menü rechts aus dem Bildschirm herausklappen, so wird versucht, es weiter nach links herausklappen zu lassen. Zunächst orientiert sich der Menü-Manager dabei am rechten Rand des Menütitels. Ist dieser auch nicht mehr im Bildschirm, so werden die Menüeinträge rechtsbündig zum rechten Rand des Bildschirms gezeichnet.

Für die vertikale Richtung gilt: Klappt ein Menü unter den unteren Rand des Bildschirms, so klappt es genau dann nach oben, wenn dadurch mehr Menüeinträge sichtbar werden würden, als wenn es nach unten klappen würde. Ein ähnlicher Algorithmus wird auch bei Windows von Microsoft benutzt. Reicht der Platz dann immer noch nicht, so wird eben am Bildschirmrand abgeschnitten.

Eventuelle Ränder, Schatten und andere Attribute (z.B. OUTLINED) werden komplett berücksichtigt, so daß die Boxen der Menüeinträge auch verschiedene Attribute haben können.

Wird auf einen grauen Menütitel geklickt, so klappt das Menü nicht herunter. Das erste Menü in einem Fenster kann nur einen Eintrag erhalten. Die Deskaccessories werden

automatisch abgetrennt, da auf die Dateinamen der Accessories einerseits nicht zugegriffen werden kann, andererseits bei späteren GEM-Versionen die Accessory-Einträge nicht mehr in die eigene Menüzeile eingetragen werden. Vielmehr werden diese kopiert, so daß auf die Accessory-Einträge nicht mehr richtig zugegriffen werden kann.

Der Menü-Manager wird beendet, wenn der Mausknopf losgelassen bzw. nochmals gedrückt wird. Dies hängt davon ab, wie der Menü-Manager betreten wurde. Dabei gilt das gleiche wie für die Pop-Up-Menüs (s.o.), d.h. es gibt eine GEM-ähnliche oder eine Macintosh-ähnliche Bedienung.

Wurde beim Verlassen außerhalb der Menüs geklickt, so passiert nichts. Ansonsten wird über die Funktion „hndl_menu" der Fensterstruktur die Routine des jeweiligen Fensters aufgerufen, welche für das Abarbeiten des Menüs zuständig ist. An diese werden dann die Parameter „window", „title" und „item" übergeben, d.h. das aktuelle Fenster und das angewählte Menü (item) eines Menütitels (title).

BOOLEAN menu_key (WINDOWP window, MKINFO *mk);

Für ein Fenster wird getestet, ob eine Taste auf ein bestimmtes Menü zutrifft. Ist dies der Fall, so wird eine entsprechende Funktion aufgerufen.

menu_key: TRUE, wenn das Fenster eine Menüzeile hat und eine Taste auf ein Menü zutrifft, FALSE sonst
window : Zeiger auf das Fenster, von welchem ein Menü über eine Taste aufgerufen werden soll
mk : Zeiger auf die Maus und Tastatur-Information beim Eintreten des Tastaturereignisses

Für das Desktop-Fenster wird das globale Menü benutzt. Für das Testen der Tastatur wird die globale Funktion „is_menu_key" des Moduls GLOBAL benutzt.

BOOLEAN init_windows (WORD err_nowindow, WORD max_reswind);

Das Modul WINDOWS wird initialisiert. Dabei wird eine Konstante für die Fehlermeldung „Keine weiteren Fenster mehr" und die Anzahl maximal zu erwartender residenter Fenster übergeben.

init_windows : TRUE, wenn die Initialisierung geklappt hat, FALSE sonst
err_nowindow: Index der Alertbox, die vom Window-Manager aufgerufen werden soll, wenn keine weiteren Fenster mehr erzeugt werden können, oder NIL, wenn keine Meldung erscheinen soll
max_reswind : Maximale Anzahl der zu erwartenden residenten Fenster

Die Initialisierung läuft schief, wenn nicht mehr genügend Speicher zur Verfügung steht, um genügend Strukturen des Typs WINDOW zur Verfügung zu stellen. Die Anzahl der maximal gleichzeitig zur Verfügung stehenden Strukturen werden in „max_reswind" übergeben. Eine Struktur bleibt resident im Speicher, wenn ein Fenster beim Schließen nicht gelöscht wird. Dies ist der Fall, wenn das Flag „WI_RESIDENT" gesetzt ist und es keine Möglichkeit gibt, das Fenster komplett zu löschen (etwa durch eine Piktogramm-Operation).

Die Anzahl errechnet sich folgendermaßen: Da in GEM maximal sieben Fenster der gleichen Klasse geöffnet werden können, können pro Klasse, bei der die Fenster resident bleiben, auch maximal sieben Strukturen im Speicher stehen. Von den Fenstern, die nicht resident bleiben, können maximal sieben gleichzeitig geöffnet werden. Da beim Schließen auch die Datenstruktur freigegeben wird, kann sie für das nächste Fenster genutzt werden. Faustregel ist also

7 * „Anzahl der residenten Klassen" + 7

Ausnahmen sind Klassen, bei denen nur maximal ein Fenster existieren kann. Dies sind z.B. der Desktop oder das Papierkorb- bzw. Klemmbrettfenster. In unserer Beispielapplikation können maximal 24 Fenster existieren (maximal sieben davon offen bzw. acht bei Berücksichtigung des Desktops):

1 Desktop-Fenster
1 Papierkorb-Fenster
1 Klemmbrett-Fenster
7 Grafik-Fenster
7 Fenster mit mathematischen Potenzen

Das sind siebzehn Fenster, die resident im Speicher gehalten werden. Dazu kommen maximal sieben weitere Fenster, die nicht resident sind, und deswegen nur in geöffnetem Zustand existieren:

7 Meta-Datei-Fenster oder
7 Bit-Image-Datei-Fenster oder
7 Text-Fenster

Letztere drei Klassen sind nicht resident, so daß eine beliebige Kombination von ihnen (zusammen maximal sieben) gleichzeitig offen sein kann.

Die Anzahl der Fenster hätte auch dynamisch ermittelt werden können, indem bei jedem Öffnen eines Fensters für eine Fensterstruktur Speicherplatz reserviert werden würde. Für ein Accessory kann dies aber böse Folgen haben, da beim Beenden einer Applikation auch der Speicherplatz des Accessories freigegeben wird, der zum Zeitpunkt des Ablaufs der Applikation angefordert wurde.

BOOLEAN term_windows (VOID);

Das Modul WINDOWS wird terminiert. Es werden alle Fenster geschlossen und die Datenstrukturen für die Fenster freigegeben.

term_windows: TRUE, wenn die Terminierung geklappt hat, FALSE sonst

### 6.5.4 Modul RCM

Das Resource-Create-Modul wurde schon im Kapitel 2 besprochen. Seine Erwähnung dient hier nur der Vollständigkeit halber.

### 6.5.5 Modul EVENT

Ein GEM-Programm wird üblicherweise durch eine Ereignisverwaltung bestimmt. Der Benutzer agiert dabei mit Hilfe der Maus oder der Tastatur, um dem Rechner bestimmte Befehle zu geben. An einer zentralen Stelle wird auf alle möglichen auftretenden Ereignisse gewartet. Tritt ein Ereignis ein, auf das die Applikation gewartet hat, so reagiert diese darauf.

In den meisten GEM-Programmen läuft diese Ereignisabfrage ähnlich ab. Über einen Aufruf der Event-Library wird mit Hilfe der Funktion „evnt_multi" oder „evnt_event" (X/GEM) auf alle möglichen Ereignisse gewartet. Nach Auftreten des Ereignisses verzweigt das Programm in verschiedene Stellen, um die Ereignisse abzuarbeiten. Ereignisse können sein: Tastaturereignisse, Mausknopfereignisse, Mausbewegungsereignisse für verschiedene Mausformen, Nachrichtenereignisse (Menüs, Fensternachrichten) und Zeitereignisse für Multitasking.

Der Event-Manager ist praktisch unabhängig von der Applikation konzipiert worden, so daß er für alle Applikationen oder Accessories benutzt werden kann.

a) Globale Variablen

GLOBAL WORD bmask;

Die Maske für die Mausknöpfe gibt an, auf welche Mausköpfe gewartet werden soll. Das niederwertigste Bit bezeichnet den am weitesten links stehenden Mausknopf. Die Variable wird in der Initialisierungsroutine des Moduls auf 1 initialisiert, d.h. es wird auf den linken Mausknopf gewartet. Leider ist es in GEM standardmäßig nicht möglich, zusätzlich auf den rechten Mauskopf zu warten, also „bmask" auf 3 zu setzen. GEM würde dann auf das gleichzeitige Drücken des linken und rechten Mausknopfes warten. Erst in X/GEM ist dieser Fehler behoben.

GLOBAL WORD bstate;

Der Status für die Mausknöpfe gibt an, ob auf das Drücken oder das Loslassen eines Mausknopfes gewartet werden soll. Dabei bedeutet ein gesetztes Bit, daß auf das Drücken gewartet werden soll. Das niederwertigste Bit bezeichnet den am weitesten links stehenden Mausknopf. Die Variable wird in der Initialisierungsroutine des Moduls auf 1 initialisiert, d.h. es wird auf das Drücken des linken Mausknopfes gewartet.

Normalerweise werden die Bits dieser Variablen jeweils umgeschaltet, so daß nach dem Warten auf das Drücken eines Mausknopfes auf dessen Loslassen gewartet wird und umgekehrt. Es kann allerdings auch sinnvoll sein, ständig auf das Drücken der Knöpfe zu warten. Dies ist etwa der Fall, wenn auf eine Funktionstaste geklickt wurde. In diesem Fall wird nochmals auf das Drücken der Knöpfe gewartet, damit beim durchgehenden Drücken des Mausknopfes ständig Mausknopfereignisse auftreten. Ein Beispiel hierfür wird im Modul DESKTOP gegeben.

b) Globale Funktionen

VOID hndl_events (VOID);

Diese Funktion handhabt alle Ereignisse, die in GEM auftreten können. Sie wird vom Hauptprogramm aufgerufen und terminiert genau dann, wenn das Programm verlassen werden soll. Bei einem Accessory wird diese Funktion niemals beendet.

Vor dem Warten auf die entsprechenden Ereignisse wird festgestellt, ob ein Fenster dieses Prozesses eine eigene Mausform hat und oben liegt. Ist dies der Fall, so wird auf den Eintritt in oder den Austritt aus dem Scrollbereich des Fensters gewartet. Hat das oberste Fenster durch Eintreten irgendeines Ereignisses gewechselt, so wird auf jeden Fall auf den Eintritt in den Bereich gewartet.

Die Funktion wartet in einer Schleife mittels „evnt_multi" auf alle eintretenden GEM-Ereignisse. Für jedes Ereignis wird eine entsprechende Routine aufgerufen, die jeweils lokal zu diesem Modul definiert ist.

Für die X/GEM-Version wird bei jedem Schleifendurchlauf festgestellt, ob sich noch ein offenes Fenster in diesem Prozeß befindet. Ist dies nicht der Fall, so wird die Applikation beendet, da die Menüzeile nach dem Schließen des letzten Fensters nicht mehr zugänglich ist.

Außerdem wird die Funktion „updt_menu" aufgerufen, die prüft, ob sich durch ein Ereignis irgendein Menü verändert hat, also z.B. abgeschaltet werden muß (siehe auch Modul MENU).

c) lokale Funktionen

VOID hndl_keybd (MKINFO *mk);

Das Ereignis „Drücken einer Taste" wird abgearbeitet.

mk: Zeiger auf die Maus- und Tastaturinformation

Beim Auftreten eines Tastaturereignisses wird allen Fenstern (von oben nach unten) die Information über einen Aufruf von „key_all" des Moduls WINDOWS angeboten. Zusätzlich wird geprüft, ob eine Menüzeile existiert, das Desktop-Fenster aber nicht vorhanden oder geschlossen ist. In diesem Fall kann die Taste nicht über das Desktopfenster in der Routine „key_all" erreicht werden. Die Funktion „is_menu_key" muß dann mit dem globalen Menü aufgerufen werden, um auf die Taste zu reagieren.

VOID hndl_button (MKINFO *mk);

Das Ereignis „Mausknopf gedrückt oder losgelassen" wird abgearbeitet.

mk: Zeiger auf die Maus- und Tastaturinformation

Wurde auf das Loslassen eines Knopfes gewartet, so wird umgeschaltet, so daß das nächste Mausknopf-Ereignis das Drücken eines Knopfes ist.

Ansonsten wird das Fenster gesucht, in welches geklickt wurde. Handelt es sich um ein Fenster dieses Prozesses, so wird zunächst geprüft, ob sich eine Menüzeile in ihm befindet. In diesem Falle wird zum Menü-Manager verzweigt und, falls in die Menüzeile geklickt wurde, werden entsprechende Routinen ausgeführt und danach die Klick-Behandlung abgebrochen.

Hat ein Fenster keine Menüzeile oder wurde nicht in die Menüzeile geklickt, so wird über „click_window" des Window-Managers das Fenster angeklickt.

Wurde kein Fenster gefunden, so wurde eventuell auf den leeren Desktop geklickt, falls die Applikation keinen Desktop angemeldet hat. In diesem Falle wird das zuletzt selektierte Fenster deselektiert. Dies funktioniert nur bei der alten Atari GEM-Version. Bei neueren Versionen erhält die Applikation kein Ereignis, wenn in den nicht angemeldeten Desktop geklickt wurde.

VOID hndl_m1 (WINDOWP top, UWORD *m1flags);

Das Ereignis „Maus ist in Fenster eingetreten oder hat dieses verlassen" wird abgearbeitet.

top : Zeiger auf das oberste Fenster
m1flags: Zeiger auf die Flags für das Eintreten in das Rechteck oder Verlassen des Rechtecks

Dieses Mausereignis wird für den Cursor benutzt, der sich im Scrollbereich eines Fensters ändern darf. Wenn „m1flags" gesetzt ist, so hat die Maus den Bereich verlassen und wird auf einen Pfeil gesetzt. Ansonsten wird sie auf die Mausform gesetzt, die in der Fensterstruktur gesetzt ist, falls es ein solches Fenster gibt (top != NULL).

Danach werden die „m1flags“ invertiert, um auf das gegenteilige Ereignis zu warten (Eintritt oder Austritt).

VOID hndl_mesag (WORD *msgbuff);

Das Ereignis „Nachricht eingetroffen“ wird abgearbeitet.

msgbuff: Zeiger auf ein Feld mit acht WORD-Werten, welches die Nachricht spezifiziert. Die Nachrichten wurden in Kapitel 2 beschrieben.

Im wesentlichen wird entweder die Menübehandlung aufgerufen (hndl_menu) oder eine entsprechende Funktion des Window-Managers.

Wird das Öffnen eines Accessories verlangt, so wird der Desktop geöffnet. Der Desktop selbst erscheint in einem Fenster. Statt des Desktops kann auch jedes andere Fenster oder eine Dialogbox geöffnet werden.

Beim Schließen eines Accessories werden alle zu diesem Prozeß gehörenden Fenster geschlossen.

VOID hndl_timer (LONG millisecs);

Das Ereignis „Zeitspanne abgelaufen“ wird abgearbeitet.

millisecs: Anzahl der Millisekunden, die seit dem letzten Zeitereignis verstrichen sind.

Die Funktion ruft „timer_all“ des Window-Managers auf, um allen Fenstern die Möglichkeit zu geben, auf das Zeitereignis zu reagieren. Je nach Fenster können unterschiedliche Dinge passieren, so daß eine Art Multitasking implementiert werden kann.

### 6.5.6 Modul INITERM

Das Modul hat die Aufgabe, alle anderen Module (außer GEMAIN) zu initialisieren und zu terminieren. Dazu stellt es zwei Funktionen zur Verfügung.

BOOLEAN init_initerm (INT argc, BYTE *argv []);

Alle Module werden initialisiert.

init_initerm: TRUE, wenn die Initialisierung geklappt hat, FALSE sonst
argc : „argc“ aus der Parameterliste von „main“
argv : „argv“ aus der Parameterliste von „main“

In dieser Funktion werden alle Module außer GEMAIN einmal aufgerufen und damit initialisiert. Die Reihenfolge der Initialisierung ist wichtig. Zuerst muß das Modul GLOBAL initialisiert werden, da dort das Anmelden beim GEM geschieht. Wenn dies nicht gelingt, wird sofort abgebrochen. Hier werden „argc" und „argv" benutzt sowie das Menü für das Accessory und die Klasse für den Desktop angegeben.

Dann folgen die Resourcen. Klappt die Initialisierung nicht, wird ebenfalls sofort abgebrochen.

Nun muß der Window-Manager initialisiert werden. An ihn wird der Index für die Fehlermeldung übergeben, die aufgerufen wird, wenn kein Fenster mehr kreiert werden kann. Außerdem wird die Anzahl der zu erwartenden residenten Fenster übergeben (siehe „init_windows").

Nun folgt die Initialisierung der anderen Module. Ihre Reihenfolge spielt keine große Rolle mehr.

Nach dem Initialisieren wird die virtuelle Workstation geöffnet. Falls es sich nicht um ein Deskaccessory handelt, wird außerdem die Menüzeile gezeichnet, falls eine solche existiert. Danach wird zunächst der Desktop, dann noch das Klemmbrettfenster geöffnet.

Falls die Kommandozeile (tail) nicht leer ist, werden noch Fenster für alle Dateien geöffnet. Handelt es sich um Bit-Image- oder Metadateien, dann werden diese angezeigt. Alle anderen Typen werden als Textdateien angezeigt.

BOOLEAN term_interm (VOID);

Alle Module werden terminiert.

term_interm: TRUE, wenn die Terminierung geklappt hat, FALSE sonst

Beim Terminieren wird festgestellt, ob die Applikation von einer anderen Applikation aufgerufen wurde, die dies durch Anhängen des Programmnamens an die Parameter kundgetan hat (siehe auch Kapitel 5).

Ist dies der Fall, so wird über einen „shel_write"-Befehl dieses Programm aufgerufen. Dabei wird angenommen, daß es sich um eine Grafik-Applikation handelt. Neben den Parametern, die vom Aufrufer gekommen sind, wird der eigene Programmname angehängt. Bei X/GEM-Applikationen ist der gesamte Vorgang nicht nötig, da die aufrufende Applikation zu diesem Zeitpunkt noch aktiv ist bzw. aktiv sein kann. Ein erneuter Aufruf würde nur eine neue Inkarnation der aufrufenden Applikation zur Folge haben.

Nun wird die Menüzeile freigegeben. Danach folgen die Terminierungsroutinen aller Module, die oben initialisiert worden sind. Dies geschieht in umgekehrter Reihenfolge. Am Ende wird das Modul GLOBAL terminiert.

### 6.5.7 Modul RESOURCE

Das Modul stellt die Funktionen zur Initialisierung der Resourcen zur Verfügung. Dazu gehört das Laden der Resource-Datei, wenn vorhanden, sowie das Einrichten von bestimmten benutzerdefinierten Objekten. Außerdem werden die Piktogramme und Bit-Images vom Standardformat in das gerätespezifische Format transformiert.

a) Globale Variablen

```
GLOBAL OBJECT *infmeta;
GLOBAL OBJECT *popup;
GLOBAL OBJECT *userimg;
GLOBAL OBJECT *icons;
GLOBAL OBJECT *settings;
GLOBAL OBJECT *sethelp;
GLOBAL OBJECT *clipmenu;
```

Es handelt sich hierbei um alle Objektbäume, die nicht im Modul GLOBAL definiert sind. Alle Module, die Objektbäume benutzen, müssen RESOURCE.H einbinden.

b) Globale Funktionen

```
BOOLEAN init_resource (VOID);
```

Das Modul wird initialisiert.

Wenn das Makro RSC_CREATE auf eins gesetzt ist, so werden die Resourcen über die Funktion „rsc_create" initialisiert. Dabei muß die Textdatei SCRAP.RSH eingebunden werden. Sie wird erzeugt, indem im Programm RCS.APP von Digital Research im Menü „Global" der Menüpunkt „Output..." angewählt und „Source file for resource" angeklickt wird. Beim Speichern der Resource-Datei wird dann SCRAP.RSH erzeugt.

Ist das Macro nicht gesetzt, so wird die Resource-Datei geladen. Für das alte GEM wird dazu der Name der Resource-Datei benötigt. Er wird in einem Makro angegeben. In jedem anderen Fall wird dieser aus dem Namen der Applikation durch Konkatenieren des Suffix „RSC" erzeugt. Beim Atari könnte der Fall eintreten, daß der Name der Applikation nicht richtig gesetzt wird, wenn eine Shell benutzt wird, die nicht „shel_write" benutzt, sondern „Pexec". Dann würde auch die Resource-Datei nicht richtig geladen werden können, wenn sich ihr Name aus dem Namen der Applikation ableitet.

Wird die Resource-Datei nicht gefunden, so wird eine Warnung ausgegeben. Dies ist die einzige Warnung, die sich nicht in einer Resource-Datei befinden kann. Aus diesem Grund ist sie nicht in einer speziellen Sprache verfaßt.

Da Accessories bei nicht vorhandener Resource-Datei nicht existieren sollten, wird das Accessory-Menü abgemeldet und endlos auf das Verstreichen einer bestimmten Zeit gewartet (jeweils ungefähr 65s). Damit kommen auch die anderen Prozesse zum Zug. Aus diesem Grund sollten Accessories keine Resource-Datei einladen, sondern RSC_CREATE benutzen.

Nun werden die Resource-Adressen der Fehlermeldungen und Bäume geholt. Danach wird eine virtuelle Workstation geöffnet, da damit das „vdi_handle" gültig ist und die Anzahl der Farben bestimmt wird. Das „vdi_handle" wird für das Transformieren der Piktogramme benötigt.

Für jeden Objektbaum wird die Routine „fix_objs" aufgerufen, die in diesem Modul lokal definiert ist und eine zentrale Funktion ausübt.

Zunächst wird jede Dialogbox in den OUTLINED-Status gesetzt. Die GEM-Funktion „rsrc_load" setzt seit den neueren GEM-Versionen die Boxen immer auf SHADOWED, auch wenn sie als OUTLINED definiert wurden.

Alle Objekte jedes Baumes werden nun durchgegangen. Falls es sich um eine Menüzeile handelt, werden für die neueren GEM-Versionen die unansehbaren Trennzeichen, die sich aus dem Zeichen ASCII 19 ergeben, durch normale Bindestriche ersetzt. Jene Zeichen werden von neueren Resource-Construction-Sets benutzt. Sie sind aber nicht symmetrisch und sehen klobig aus. Wahrscheinlich mußte Digital Research diese ändern, weil sie zu sehr nach den Macintosh-Trennzeichen ausgesehen haben.

Handelt es sich um Piktogramme, so werden diese transformiert. Zusätzlich wird der Rahmen, der sich um die Piktogramme legt, also die Objekthöhe, aus den Werten des Piktogrammes errechnet. Damit kommt es nicht mehr zu unerwünschten falschen Objekthöhen bei niedriger Auflösung. Sie haben sich dadurch bemerkbar gemacht, daß die Objekthöhe kleiner als die Piktogrammhöhe war.

Wird ein erweiterter Typ gefunden, so werden die nötigen Initialisierungen für die benutzerdefinierten Objekte gemacht. Durch die Benutzung verschiedener Auflösungen ist es eine nicht gerade triviale Aufgabe, die ankreuzbaren Boxen quadratisch und die Radio-Buttons rund zu halten. Dabei wird mehr Maß auf Ästhetik beim Algorithmus des Linienzeichnens gelegt (regelmäßige Treppen), als auf das exakte Verhältnis der Rechteckseiten.

Ist ein Desktop in der Resource-Datei vorhanden, so werden verschiedene Initialisierungen durchgeführt. Die Infozeile des Desktop wird versteckt und auf leer gesetzt. Ist die Breite des Bildschirms kleiner als die Breite des Desktops (z.B. 640 Pixel), so wird der Desktop in ein Fenster gelegt. Dazu wird die Klasse des Desktops in „DESKWINDOW" umgewandelt. Für den Atari ST wird der Desktop in mittlerer Auflösung in Grün dargestellt, wie dies üblich ist. Ansonsten wird er in Blau dargestellt.

Nun wird die Objektbreite des Desktop-Baums angepaßt, falls der Bildschirm größer ist. Dadurch ist das Programm auch auf einem Großbildschirm lauffähig. Schließlich werden die Funktionstasten und Piktogramme entsprechend plaziert.

BOOLEAN term_resource (VOID);

Das Modul wird terminiert. Beim Terminieren werden die Resourcen freigegeben.

term_resource: TRUE, wenn die Terminierung geklappt hat, FALSE sonst

### 6.5.8 Modul MENU

Das Modul behandelt die Menüverwaltung für die globale Menüzeile. Falls es sich bei der Applikation um ein Accessory handelt, bietet das Modul auch die Behandlung der Menüzeile im Fenster des Accessories.

a) Globale Variablen

GLOBAL BOOLEAN menu_ok;

Die Variable ist TRUE, wenn der Menübaum nicht NULL, also vorhanden ist, FALSE sonst.

GLOBAL BOOLEAN menu_fits;

Die Variable ist TRUE, wenn die Menüzeile vorhanden ist und auf den Desktop paßt, FALSE sonst. Bei niedriger Auflösung ist die Variable also FALSE, wenn die Menüzeile mehr als 40 Zeichen umfaßt.

GLOBAL FUNCINFO funcmenus [MAX_FUNC];

Der Typ FUNCINFO besteht aus den Komponenten „title" und „item". Für jede Funktionstaste wird in dieser Variablen abgelegt, welcher Menütitel und welcher Menüeintrag bei Betätigen der Funktionstaste ausgelöst wird. Dieses Feld wird nur benötigt, wenn Funktionstasten auch auf dem Desktop sichtbar sind. Ansonsten genügt es, die Funktionalität in den Menüeinträgen festzulegen (siehe auch Kapitel 5).

GLOBAL SET menus;

Diese Menge gibt die Nummern der Menüs an, die gerade aktiv sind.

b) Globale Funktionen

VOID updt_menu (WINDOWP window);

Die Menüeinträge werden auf den neuesten Stand gebracht, d.h. sie werden ein- oder ausgeschaltet (enabled oder disabled). Die Funktion wird nach jedem Ereignis aus dem Modul EVENT aufgerufen.

window: Zeiger auf das Fenster, in welchem sich die Menüzeile befindet oder NULL, wenn es sich um die globale GEM-Menüzeile handelt

Da die Menüzeile auch in einem Fenster liegen kann, muß der Parameter „window" mit angegeben werden. Die Funktion kann nämlich auch vom Window-Manager aufgerufen werden, der immer als Parameter das Fenster übergibt.

Die einzelnen Menüeinträge werden abhängig vom Zustand der Applikation ein- oder ausgeschaltet. Falls Funktionstasten auf dem Desktop sichtbar sind, so werden diese an die neue Situation angepaßt. Nur wenn sich etwas geändert hat, werden diese neu gezeichnet.

VOID hndl_menu (WINDOWP window, WORD title, WORD item);

Die Funktion übernimmt die Abarbeitung der globalen Menüzeile bzw. der Menüzeile des Desktops, falls dieser in einem Fenster liegt.

window: Zeiger auf das Fenster, in welchem sich die Menüzeile befindet oder NULL, wenn es sich um die globale GEM-Menüzeile handelt
title : Titel des ausgewählten Menüs
item : Eintrag des ausgewählten Menüs

Da die Menüzeile auch in einem Fenster liegen kann, muß der Parameter „window" mit angegeben werden. Bei einem Ereignis „Menü ausgewählt", welches von GEM geliefert wird, muß als Parameter „window" = NULL übergeben werden.

Je nach Menütitel und Menüeintrag wird zu einer entsprechenden Funktion verzweigt. Da die meisten Menüs allerdings allgemeingültig sind, muß keine spezielle Funktion aufgerufen werden.

Falls das Menü „Über..." aufgerufen wird, wird zur Funktion „mabout" verzweigt, für das Menü „Einstellungen..." zur Funktion „msettings". Die beiden Funktionen sind in diesem Modul definiert.

BOOLEAN init_menu (VOID);

Das Modul wird initialisiert.

init_menu: TRUE, wenn die Initialisierung geklappt hat, FALSE sonst

Beim Initialisieren werden zunächst die Menüs der Funktionstasten der Variablen „funcmenus" zugewiesen.

Daraufhin wird festgestellt, ob die Menüzeile auf den Bildschirm paßt. Ist dies nicht der Fall, so wird die Klasse des Desktops auf DESKWINDOW gesetzt, so daß die Menüzeile nun in einem Fenster erscheint.

Ist die Menüzeile vorhanden, so wird der Name des Aufrufers in den vorletzten Eintrag des Menüs „Datei" eingesetzt. Dies ist für Programme wie z.B. OUTPUT notwendig, damit diese ihren Aufrufer wieder starten können. Falls kein Aufrufer existiert, wird der Menüeintrag (MCALLER) gelöscht.

Wenn die Menüzeile normal auf dem Bildschirm erscheint, so wird getestet, ob ein Drop-Down-Menü aus dem rechten Rand des Bildschirms herausklappt. Falls dies der Fall ist, wird das Menü entsprechend korrigiert. Man muß sich also keine Gedanken mehr über die Breite seiner Menüs machen.

BOOLEAN term_menu (VOID);

Das Modul wird terminiert.

term_menu: TRUE, wenn die Terminierung geklappt hat, FALSE sonst

### 6.5.9 Modul DESKTOP

Das Modul zeigt ein Beispiel für einen eigenen Desktop mit einigen Piktogrammen. Er wurde ähnlich angelegt wie der GEM-Desktop. Allerdings befinden sich in ihm noch Funktionstasten. Der Desktop wird bei X/GEM in ein Fenster gelegt, bei anderen Versionen ersetzt er den von GEM bekannten „grau gemusterten Desktop".

WINDOWP find_desk (VOID);

Die Funktion liefert einen Zeiger auf die Beschreibung des Desktop-Fensters.

find_desk: Zeiger auf das Fenster, das den Desktop beschreibt, oder NULL, wenn kein
Desktop existiert.

VOID get_dxywh (WORD obj, RECT *border);

Das Rechteck, das ein Objekt im Desktop gerade einschließt (z.B. eines Piktogramms) kann im Desktop ermittelt werden.

obj : Index des Objekts
border : Umrandung des Objekts

Ist das Desktop-Fenster nicht vorhanden oder geschlossen, kein Objektbaum in ihn eingeklinkt oder „obj" NIL, so wird ein Rechteck ohne Breite und Höhe (jeweils null) zurückgegeben. Die Koordinaten sind dann ebenfalls null.

VOID set_func (CONST BYTE *keys);

Die Funktionstasten des Desktops können beschriftet werden. Sie werden nicht sofort gezeichnet. Dies muß mit „draw_func" oder „draw_key" (s.u.) erledigt werden.

keys: Zeiger auf die Zeichenkette, welche die Funktionstasten beschreibt

Die Texte der einzelnen Funktionstasten werden durch Kommata getrennt. Sollen bestimmte Funktionstasten erhalten bleiben, so kann statt des Eintrags auch ein $-Zeichen auftreten. So werden z.B. durch die Zeichenkette

„Öffnen,Schließen,$,$,$,$,Ausschneiden,Kopieren,Einfügen,Ende"

die Funktionstasten 1, 2 und 7 bis 10 beschriftet, die Tasten 3 bis 6 behalten jeweils ihren alten Wert. Es sind nur die ersten 8 Zeichen jeder Tastenbeschriftung signifikant.

VOID draw_func (VOID);

Die Funktionstasten werden auf dem Desktops gezeichnet.

VOID draw_key (WORD key);

Eine einzelne von 10 Funktionstasten wird gezeichnet. Die Funktionstasten werden von eins beginnend durchnumeriert.

key: Nummer der Funktionstaste (1 – 10)

VOID set_deskinfo (CONST BYTE *info, BOOLEAN center);

Die Infozeile des Desktops wird besetzt.

info : Zeiger auf die Zeichenkette, die in die Infozeile gesetzt werden soll
center : TRUE, wenn die Zeichenkette zentriert werden soll, FALSE sonst

Befindet sich der Desktop in einem Fenster, so wird die Infozeile des Fensters gesetzt. In diesem Fall wird der Parameter „center" nicht berücksichtigt, sondern die Infozeile von links beginnend ausgegeben.

VOID set_meminfo (VOID);

Die Speicherinformation wird angezeigt. Sie befindet sich in unserer Beispielapplikation in der rechten unteren Ecke. Es handelt sich um die Anzahl des noch verfügbaren freien Hauptspeichers in Kilobytes.

WINDOWP crt_desktop (OBJECT *obj, OBJECT *menu, WORD icon);

Das Desktopfenster wird kreiert.

crt_desktop: Zeiger auf das Fenster oder NULL, wenn es nicht kreiert werden konnte
obj : Zeiger auf den Objektbaum, welcher sich im Fenster befinden soll, also der eigene Desktop
menu : Menüzeile, welche sich im Desktopfenster befinden soll, oder NULL, wenn sich nur die globale Menüzeile am oberen Rand des Bildschirms befindet
icon : Index des Piktogramms, aus dem das Fenster entstehen soll, oder NIL, wenn es kein solches Piktogramm gibt

Beim Kreieren wird berücksichtigt, ob die Klasse des Desktops DESK oder DESKWINDOW ist, es sich also um den Desktop handelt, welcher den GEM-Desktop ersetzt, oder um einen Desktop, der sich in einem Fenster befindet. Letzteres ist bei X/GEM der Fall oder bei Accessories, welche einen eigenen Desktop benötigen (man denke an WORDPLUS, das als Accessory laufen soll).

Die Werte der Fensterstruktur werden hier eingesetzt. Die Faktoren XFAC und YFAC sind jeweils zwei, da der Desktop minimal um zwei Pixel je Richtung gescrollt werden darf. Sonst gibt es Probleme beim Zeichnen des „grau gemusterten" Desktop-Hintergrundes.

Die beiden Komponenten „hndl_menu" und „updt_menu" des Desktops werden auf die entsprechenden globalen Funktionen des Moduls MENU gesetzt. Der Grund dafür ist darin zu suchen, daß es sich bei der Menüzeile des Desktops meistens um die globale GEM-Menüzeile handelt. Selbst wenn dies nicht der Fall ist (Accessories mit eigener Menüzeile), werden diese richtig gehandhabt, da die Parameter der beiden globalen Funktionen korrekt angegeben wurden (siehe auch Modul MENU). Damit kann auch der integrierte Menü-Manager, der die Menüzeilen in Fenstern verwaltet, die beiden Funktionen richtig aufrufen.

Liegt der Desktop in einem Fenster und ist ein Objektbaum vorhanden, so muß die Dokumentgröße angepaßt werden, wenn die Objektbaumbreite oder -höhe größer als die des physikalischen Desktops ist. Ist kein Objektbaum vorhanden, so werden immer große Schieber gezeigt.

Existiert ein Objektbaum, so wird das Fenster so positioniert, daß der untere Teil des Desktops gezeigt wird. In unserem Fall ist dies günstig, da sich dort die Funktionstasten befinden.

BOOLEAN open_desktop (WORD icon);

Das Desktopfenster wird geöffnet.

open_desktop: TRUE, wenn das Fenster geöffnet werden konnte, FALSE sonst
icon : Index des Piktogramms, aus dem das Fenster entstehen soll, oder NIL, wenn es kein solches Piktogramm gibt

Beim Öffnen des Desktops wird zunächst festgestellt, ob es sich um einen Desktop handelt, der den Desktop des GEM ersetzt. Ist dies der Fall, so wird das Flag HIDETREE der Infozeile im Objektbaum (DESKINFO) gelöscht, damit dieses sichtbar ist. Im anderen Fall (Desktop im Fenster) wird versucht, die globale Menüzeile in das Desktopfenster zu legen. Dies geschieht aber nur, wenn die Menüzeile entweder nicht auf den normalen Bildschirm paßt (zu breit ist) oder es sich um ein Deskaccessory handelt.

Dann wird festgestellt, ob bereits ein Desktop kreiert wurde, da es nur maximal einen Desktop geben kann. Ist dies nicht der Fall, so wird ein Desktop über die Funktion „crt_desktop" kreiert.

Hat das Kreieren geklappt oder existiert bereits ein Desktop, so wird dieser durch ein „open_window" geöffnet, wenn er noch nicht offen war. Ansonsten wird er durch ein „top_window" nach oben gebracht.

BOOLEAN info_desktop (WINDOWP window, WORD icon);

Die Information des Desktops wird angezeigt. Es handelt sich hier um dieselbe Information wie beim Auswählen des Menüs „Über...".

info_desktop : TRUE, wenn es Informationen gibt, FALSE sonst
window : Zeiger auf das Fenster, von dem Informationen gezeigt werden sollen
icon : Index des Piktogramms, von dem Informationen gezeigt werden sollen

Die Parameter „window" und „icon" sind hier ohne Bedeutung und nur der Vollständigkeit der Schnittstelle halber mit aufgenommen worden. Sie werden genau dann benötigt, wenn man zu verschiedenen Inkarnationen derselben Fensterklasse verschiedene Informationen zeigen will. Dann zeigt „window" genau auf das entsprechende Fenster. Ist dieses nicht bekannt, aber das Piktogramm, aus dem das Fenster entstanden ist, so kann man auch dieses angeben. Über ein „search_window" kann man dann das eindeutige zugehörige Fenster finden. Im Modul EDIT wird ein Beispiel für diese Parameter gegeben.

BOOLEAN help_desktop (WINDOWP window, WORD icon);

Eine Hilfemeldung für den Desktop wird angezeigt. In unserem Beispielprogramm haben wir keine Hilfemeldung eingebaut.

help_desktop: TRUE, wenn es eine Hilfemeldung gibt, FALSE sonst
window : Zeiger auf das Fenster, von dem eine Hilfemeldung gezeigt werden soll
icon : Index des Piktogramms, von dem eine Hilfemeldung gezeigt werden soll

Die Parameter „window" und „icon" sind hier ohne Bedeutung und nur wegen der Vollständigkeit der Schnittstelle halber mit aufgenommen worden. Sie werden genau dann benötigt, wenn man zu verschiedenen Inkarnationen derselben Fensterklasse verschiedene Hilfemeldungen benötigt. Dann zeigt „window" auf das entsprechende Fenster. Ist dieses nicht bekannt, aber das Piktogramm, aus dem das Fenster entstanden ist, so kann man auch dieses angeben. Über ein „search_window" kann man dann das eindeutige zugehörige Fenster finden.

BOOLEAN init_desktop (VOID);

Das Modul wird initialisiert.

init_desktop: TRUE, wenn die Initialisierung geklappt hat, FALSE sonst

BOOLEAN term_desktop (VOID);

Das Modul wird terminiert.

term_desktop: TRUE, wenn die Terminierung geklappt hat, FALSE sonst

Im Desktop gibt es eine Menge lokaler Funktionen, auf die aber hier nicht näher eingegangen werden soll. Wir verweisen nur auf den kommentierten Quelltext, der auf Diskette vorhanden ist. Ähnliches gilt auch für die weiteren Module. Die Kapazität des Buches würde leicht gesprengt werden, wenn die lokalen Funktionen aller Module in allen Details erklärt werden würden.

### 6.5.10 Modul CLIPBRD

Das Modul verwaltet das Klemmbrett. Dieses wird ähnlich verwaltet wie ein Fenster im GEM-Desktop, d.h. alle Dateien werden angezeigt und können durch Doppelklick geöffnet werden. In diesem Modul sind außerdem Funktionen zu finden, welche die fehlenden Scrap-Funktionen aus GEM/3 im alten GEM nachbildet.

WORD scrap_read (BYTE *pscrap);

Die Funktion ersetzt die nicht richtig arbeitende Funktion „srcp_read" der Versionen von GEM, die vor GEM/3 auf den Markt kamen.

scrap_read: Bitkombination der Standarddateien, die sich im GEM-Clipboard befinden, oder −1, wenn kein Clipboard-Pfad gesetzt ist (siehe Beschreibung „scrp_read" in Kapitel 2).
pscrap : Zeiger auf die Zeichenkette, die nach dem Aufruf den aktuellen Clipboard-Pfad enthält

WORD scrap_write (BYTE *pscrap);

Die Funktion hat die gleiche Wirkung wie „scrp_write" des AES und ist nur der Vollständigkeit halber aufgenommen worden.

scrap_write: Funktionswert der AES-Funktion „scrp_write"
pscrap : Zeiger auf die Zeichenkette, die den neuen GEM-Clipboard-Pfad enthält

WORD scrap_clear (VOID);

Die Funktion ersetzt die fehlende Funktion „srcp_clear" der Versionen von GEM, die vor GEM/3 auf den Markt kamen.

scrap_clear: 0, wenn ein Fehler aufgetreten ist, > 0 sonst

Es werden alle Dateien SCRAP.* gelöscht. Für neuere GEM-Versionen wird die eingebaute Funktion „scrp_clear" benutzt.

VOID get_clipxywh (WORD obj, RECT *border);

Das Rechteck eines Objekts (z.B. eines Piktogramms) im Clipboard-Fenster kann ermittelt werden.

obj : Index des Objekts
border : Umrandung des Objekts

Ist das Clipboard-Fenster nicht vorhanden oder geschlossen oder „obj" NIL, so wird ein Rechteck ohne Breite und Höhe (jeweils null) zurückgegeben. Die Koordinaten sind dann ebenfalls null.

VOID print_clipfiles (WINDOWP window, SET objs);

Die Dateien des Klemmbretts werden ausgedruckt.

window: Zeiger auf das Clipboard-Fenster
objs : Zeiger auf die Menge der Objekte, die ausgedruckt werden soll

BOOLEAN icons_clipbrd (WORD src_obj, WORD dest_obj);

Operationen für das Klemmbrettpiktogramm werden durchgeführt. Diese Funktion wird vom Desktop aufgerufen, wenn das Klemmbrettpiktogramm auf ein anderes Piktogramm verschoben wird.

icons_clipbrd: TRUE, wenn es eine gültige Piktogramm-Operation gab, FALSE sonst
src_obj : Index des Quellobjektes
dest_obj : Index des Zielobjektes

Je nach Objekttyp wird eine entsprechende Operation durchgeführt. Beim Ziehen des Klemmbretts auf den Papierkorb wird ein „scrap_clear" aufgerufen. Wird es auf den Drucker gezogen, werden alle Objekte ausgedruckt.

WINDOWP crt_clipbrd (OBJECT *obj, OBJECT *menu, WORD icon);

Ein Klemmbrettfenster wird kreiert.

crt_clipbrd: Zeiger auf das kreierte Fenster oder NULL, wenn es nicht kreiert werden konnte
obj : Zeiger auf den Objektbaum, welcher sich im Fenster befinden soll
menu : Menüzeile, welche sich im Fenster befinden soll, oder NULL, wenn es keine Menüzeile geben soll
icon : Index des Piktogramms, aus dem das Fenster entstehen soll, oder NIL, wenn es kein solches Piktogramm gibt

Beim Kreieren werden die schon existierenden Fenster der gleichen Klasse gezählt. Neue Fenster dieser Klasse öffnen sich dann leicht versetzt, so daß sie sich nicht vollständig verdecken.

Die Dokumentbreite wird abhängig davon gesetzt, wie der Schalter im Klemmbrettmenü steht, welcher angibt, ob die Dateien als Piktogramme (zweidimensional) oder als Text (eindimensional) angezeigt werden sollen.

BOOLEAN open_clipbrd (WORD icon);

Ein Klemmbrettfenster wird geöffnet.

open_clipbrd: TRUE, wenn das Fenster geöffnet werden konnte, FALSE sonst
icon : Index des Piktogramms, aus dem das Fenster entstehen soll, oder NIL, wenn es kein solches Piktogramm gibt

Beim Öffnen des Klemmbretts wird zunächst festgestellt, ob es bereits ein offenes Klemmbrettfenster gibt. Falls dies der Fall ist, wird es nach oben gebracht.

Im anderen Fall wird festgestellt, ob es bereits ein geschlossenes (kreiertes) Klemmbrettfenster gibt, da in dieser Applikation nur maximal ein Klemmbrettfenster geöffnet werden kann. Ist dies nicht der Fall, so wird ein Fenster über die Funktion „crt_clipbrd" kreiert.

Hat das Kreieren geklappt oder existiert bereits ein Fenster, so wird dieses durch ein „open_window" geöffnet.

BOOLEAN info_clipbrd (WINDOWP window, WORD icon);

Die Information des Klemmbretts wird angezeigt.

info_clipbrd: TRUE, wenn es Informationen gibt, FALSE sonst
window : Zeiger auf das Fenster, von dem Informationen gezeigt werden sollen
icon : Index des Piktogramms, von dem Informationen gezeigt werden sollen

Zunächst wird der Parameter „icon" ausgewertet. Ist er ungleich NIL, so handelt es sich um ein ganz bestimmtes Fenster, das aus einem ganz bestimmten Piktogramm erzeugt wurde. Dieses Fenster wird nun über „search_window" gesucht.

Wurde ein solches Fenster gefunden oder war kein Piktogramm angegeben (NIL), dafür aber ein Fenster, so wird das Menü „mclipinfo" aufgerufen. Es befindet sich lokal in diesem Modul und kann auch über das Menü des Fensters aufgerufen werden.

BOOLEAN help_clipbrd (WINDOWP window, WORD icon);

Eine Hilfemeldung für das Klemmbrettfenster oder -piktogramm wird angezeigt.

help_clipbrd: TRUE, wenn es eine Hilfemeldung gibt, FALSE sonst
window : Zeiger auf das Fenster, von dem eine Hilfemeldung gezeigt werden soll
icon : Index des Piktogramms, von dem eine Hilfemeldung gezeigt werden soll

Für die Parameter gilt das bei „help_desktop" Gesagte.

BOOLEAN init_clipbrd (VOID);

Das Modul wird initialisiert.

init_clipbrd: TRUE, wenn die Initialisierung geklappt hat, FALSE sonst

BOOLEAN term_clipbrd (VOID);

Das Modul wird terminiert.

term_clipbrd: TRUE, wenn die Terminierung geklappt hat, FALSE sonst

### 6.5.11 Modul DISK

Das Modul bietet eine Verwaltung des Diskettenpiktogramms im Desktop. Es hat in unserer Applikation allerdings keine Bedeutung.

WINDOWP crt_disk (OBJECT *obj, OBJECT *menu, WORD icon);

Ein Diskettenfenster wird kreiert.

crt_disk: Zeiger auf das kreierte Fenster oder NULL, wenn es nicht kreiert werden konnte
obj : Zeiger auf den Objektbaum, welcher sich im Fenster befinden soll
menu : Menüzeile, welche sich im Fenster befinden soll, oder NULL, wenn es keine Menüzeile geben soll
icon : Index des Piktogramms, aus dem das Fenster entstehen soll, oder NIL, wenn es kein solches Piktogramm gibt

BOOLEAN open_disk (WORD icon);

Ein Diskettenfenster wird geöffnet.

open_disk: TRUE, wenn das Fenster geöffnet werden konnte, FALSE sonst
icon : Index des Piktogramms, aus dem das Fenster entstehen soll, oder NIL, wenn es kein solches Piktogramm gibt

BOOLEAN info_disk (WINDOWP window, WORD icon);

Die Information des Diskettenfensters oder Piktogramms wird angezeigt.

info_disk : TRUE, wenn es Informationen gibt, FALSE sonst
window : Zeiger auf das Fenster, von dem Informationen gezeigt werden sollen
icon : Index des Piktogramms, von dem Informationen gezeigt werden sollen

BOOLEAN help_disk (WINDOWP window, WORD icon);

Eine Hilfemeldung für das Diskettenfenster oder -piktogramm wird angezeigt.

help_disk : TRUE, wenn es eine Hilfemeldung gibt, FALSE sonst
window : Zeiger auf das Fenster, von dem eine Hilfemeldung gezeigt werden soll
icon : Index des Piktogramms, von dem eine Hilfemeldung gezeigt werden soll

Für die Parameter gilt das bei „help_desktop" Gesagte.

BOOLEAN init_disk (VOID);

Das Modul wird initialisiert.

jnit_disk : TRUE, wenn die Initialisierung geklappt hat, FALSE sonst

BOOLEAN term_disk (VOID);

Das Modul wird terminiert.

term_disk : TRUE, wenn die Terminierung geklappt hat, FALSE sonst

### 6.5.12 Modul PRINTER

Das Modul bietet eine Verwaltung des Druckerpiktogramms im Desktop. Es hat in unserer Applikation allerdings keine Bedeutung. Es dient lediglich anderen Piktogrammen als Ziel für die Ziehbewegung.

WINDOWP crt_printer (OBJECT *obj, OBJECT *menu, WORD icon);

Ein Druckerfenster wird kreiert.

crt_printer : Zeiger auf das kreierte Fenster oder NULL, wenn es nicht kreiert werden konnte
obj : Zeiger auf den Objektbaum, welcher sich im Fenster befinden soll
menu : Menüzeile, welche sich im Fenster befinden soll, oder NULL, wenn es keine Menüzeile geben soll
icon : Index des Piktogramms, aus dem das Fenster entstehen soll, oder NIL, wenn es kein solches Piktogramm gibt

BOOLEAN open_printer (WORD icon);

Ein Druckerfenster wird geöffnet.

open_printer: TRUE, wenn das Fenster geöffnet werden konnte, FALSE sonst
icon : Index des Piktogramms, aus dem das Fenster entstehen soll, oder NIL, wenn es kein solches Piktogramm gibt

BOOLEAN info_printer (WINDOWP window, WORD icon);

Die Information des Druckerfensters oder Piktogramms wird angezeigt.

info_printer : TRUE, wenn es Informationen gibt, FALSE sonst
window : Zeiger auf das Fenster, von dem Informationen gezeigt werden sollen
icon : Index des Piktogramms, von dem Informationen gezeigt werden sollen

BOOLEAN help_printer (WINDOWP window, WORD icon);

Eine Hilfemeldung für das Druckerfenster oder -piktogramm wird angezeigt.

help_printer : TRUE, wenn es eine Hilfemeldung gibt, FALSE sonst
window : Zeiger auf das Fenster, von dem eine Hilfemeldung gezeigt werden soll
icon : Index des Piktogramms, von dem eine Hilfemeldung gezeigt werden soll

Für die Parameter gilt das bei „help_desktop" Gesagte.

BOOLEAN init_printer (VOID);

Das Modul wird initialisiert.

init_printer : TRUE, wenn die Initialisierung geklappt hat, FALSE sonst

BOOLEAN term_printer (VOID);

Das Modul wird terminiert.

term_printer: TRUE, wenn die Terminierung geklappt hat, FALSE sonst

### 6.5.13 Modul TRASH

Das Modul bietet eine Verwaltung des Papierkorbpiktogramms im Desktop. Es hat in unserer Applikation allerdings keine Bedeutung, außer daß es geöffnet werden kann. Es dient außerdem anderen Piktogrammen als Ziel für die Ziehbewegung.

WINDOWP crt_trash (OBJECT *obj, OBJECT *menu, WORD icon);

Ein Papierkorbfenster wird kreiert.

crt_trash : Zeiger auf das kreierte Fenster oder NULL, wenn es nicht kreiert werden konnte
obj : Zeiger auf den Objektbaum, welcher sich im Fenster befinden soll
menu : Menüzeile, welche sich im Fenster befinden soll, oder NULL, wenn es keine Menüzeile geben soll
icon : Index des Piktogramms, aus dem das Fenster entstehen soll, oder NIL, wenn es kein solches Piktogramm gibt

BOOLEAN open_trash (WORD icon);

Ein Papierkorbfenster wird geöffnet.

open_trash : TRUE, wenn das Fenster geöffnet werden konnte, FALSE sonst
icon : Index des Piktogramms, aus dem das Fenster entstehen soll, oder NIL, wenn es kein solches Piktogramm gibt

Es wird maximal ein Papierkorbfenster erzeugt. Aus diesem Grund läuft der Algorithmus genauso ab wie der Algorithmus beim Öffnen des Klemmbrettpiktogramms.

BOOLEAN info_trash (WINDOWP window, WORD icon);

Die Information des Papierkorbfensters oder Piktogramms wird angezeigt.

info_trash : TRUE, wenn es Informationen gibt, FALSE sonst
window : Zeiger auf das Fenster, von dem Informationen gezeigt werden sollen
icon : Index des Piktogramms, von dem Informationen gezeigt werden sollen

BOOLEAN help_trash (WINDOWP window, WORD icon);

Eine Hilfemeldung für das Papierkorbfenster oder -piktogramm wird angezeigt.

help_trash : TRUE, wenn es eine Hilfemeldung gibt, FALSE sonst
window : Zeiger auf das Fenster, von dem eine Hilfemeldung gezeigt werden soll
icon : Index des Piktogramms, von dem eine Hilfemeldung gezeigt werden soll

Für die Parameter gilt das bei „help_desktop" Gesagte.

BOOLEAN init_trash (VOID);

Das Modul wird initialisiert.

init_trash : TRUE, wenn die Initialisierung geklappt hat, FALSE sonst

Beim Initialisieren wird sofort ein Papierkorbfenster kreiert, da dann immer eine Information möglich ist, auch wenn noch kein Fenster geöffnet ist.

BOOLEAN term_trash (VOID);

Das Modul wird terminiert.

term_trash : TRUE, wenn die Terminierung geklappt hat, FALSE sonst

### 6.5.14 Modul IMAGE

Das Modul bietet die Darstellung von Bit-Image-Dateien in einem Fenster.

```
WINDOWP crt_image (OBJECT *obj, OBJECT *menu, WORD icon,
                   BYTE *filename);
```

Ein Fenster zur Darstellung einer Bit-Image-Datei wird kreiert.

| | |
|---|---|
| crt_image | : Zeiger auf das kreierte Fenster oder NULL, wenn es nicht kreiert werden konnte |
| obj | : Zeiger auf den Objektbaum, welcher sich im Fenster befinden soll |
| menu | : Menüzeile, welche sich im Fenster befinden soll, oder NULL, wenn es keine Menüzeile geben soll |
| icon | : Index des Piktogramms, aus dem das Fenster entstehen soll, oder NIL, wenn es kein solches Piktogramm gibt |
| filename | : Zeiger auf den Namen der Bit-Image-Datei |

Ist kein Pfad im Dateinamen angegeben, so wird der aktuelle Pfad mit aufgenommen.

BOOLEAN open_image (WORD icon, BYTE *filename);

Ein Fenster zur Darstellung einer Bit-Image-Datei wird geöffnet.

| | |
|---|---|
| open_image | : TRUE, wenn das Fenster geöffnet werden konnte, FALSE sonst |
| icon | : Index des Piktogramms, aus dem das Fenster entstehen soll, oder NIL, wenn es kein solches Piktogramm gibt |
| filename | : Zeiger auf den Namen der Bit-Image-Datei |

Wenn ein ganz bestimmtes Piktogramm, z.B. aus dem Klemmbrettfenster, gemeint ist, so wird dieses nach oben gebracht, wenn es bereits offen ist. Ansonsten wird ein neues Fenster kreiert, wenn nicht bereits ein geschlossenes Fenster vorhanden ist. Dieses Fenster wird dann geöffnet.

BOOLEAN info_image (WINDOWP window, WORD icon);

Die Information des Bit-Image-Fensters wird angezeigt.

info_image : TRUE, wenn es Informationen gibt, FALSE sonst
window : Zeiger auf das Fenster, von dem Informationen gezeigt werden sollen
icon : Index des Piktogramms, von dem Informationen gezeigt werden sollen

BOOLEAN help_image (WINDOWP window, WORD icon);

Eine Hilfemeldung für das Bit-Image-Fenster wird angezeigt.

help_image: TRUE, wenn es eine Hilfemeldung gibt, FALSE sonst
window : Zeiger auf das Fenster, von dem eine Hilfemeldung gezeigt werden soll
icon : Index des Piktogramms, von dem eine Hilfemeldung gezeigt werden soll

Für die Parameter gilt das bei „help_desktop" Gesagte.

BOOLEAN init_image (VOID);

Das Modul wird initialisiert.

init_image : TRUE, wenn die Initialisierung geklappt hat, FALSE sonst

BOOLEAN term_image (VOID);

Das Modul wird terminiert.

term_image: TRUE, wenn die Terminierung geklappt hat, FALSE sonst

### 6.5.15 Modul META

Das Modul bietet die Darstellung von Metadateien in einem Fenster.

```
WINDOWP crt_meta (OBJECT *obj, OBJECT *menu, WORD icon,
                  BYTE *filename);
```

Ein Fenster zur Darstellung einer Metadatei wird kreiert.

crt_meta : Zeiger auf das kreierte Fenster oder NULL, wenn es nicht kreiert werden konnte
obj : Zeiger auf den Objektbaum, welcher sich im Fenster befinden soll
menu : Menüzeile, welche sich im Fenster befinden soll, oder NULL, wenn es keine Menüzeile geben soll
icon : Index des Piktogramms, aus dem das Fenster entstehen soll, oder NIL, wenn es kein solches Piktogramm gibt
filename : Zeiger auf den Namen der Metadatei

Ist kein Pfad im Dateinamen angegeben, so wird der aktuelle Pfad mit aufgenommen.

BOOLEAN open_meta (WORD icon, BYTE *filename);

Ein Fenster zur Darstellung einer Metadatei wird geöffnet.

open_meta : TRUE, wenn das Fenster geöffnet werden konnte, FALSE sonst
icon : Index des Piktogramms, aus dem das Fenster entstehen soll, oder NIL, wenn es kein solches Piktogramm gibt
filename : Zeiger auf den Namen der Metadatei

Wenn ein ganz bestimmtes Piktogramm, z.B. aus dem Klemmbrettfenster, gemeint ist, so wird dieses nach oben gebracht, wenn es bereits offen ist. Ansonsten wird ein neues Fenster kreiert, wenn nicht bereits ein geschlossenes Fenster vorhanden ist. Dieses Fenster wird dann geöffnet.

BOOLEAN info_meta (WINDOWP window, WORD icon);

Die Information der Metadatei wird angezeigt.

info_meta : TRUE, wenn es Informationen gibt, FALSE sonst
window : Zeiger auf das Fenster, von dem Informationen gezeigt werden sollen
icon : Index des Piktogramms, von dem Informationen gezeigt werden sollen

BOOLEAN help_meta (WINDOWP window, WORD icon);

Eine Hilfemeldung für die Metadatei wird angezeigt.

help_meta : TRUE, wenn es eine Hilfemeldung gibt, FALSE sonst
window : Zeiger auf das Fenster, von dem eine Hilfemeldung gezeigt werden soll
icon : Index des Piktogramms, von dem eine Hilfemeldung gezeigt werden soll

Für die Parameter gilt das bei „help_desktop“ Gesagte.

BOOLEAN init_meta (VOID);

Das Modul wird initialisiert.

init_meta : TRUE, wenn die Initialisierung geklappt hat, FALSE sonst

BOOLEAN term_meta (VOID);

Das Modul wird terminiert.

term_meta : TRUE, wenn die Terminierung geklappt hat, FALSE sonst

### 6.5.16 Modul EDIT

Das Modul bietet die Darstellung von editierbaren ASCII-Dateien in einem Fenster. Die Routinen zum eigentlichen Editieren sind aber noch nicht eingebaut, so daß die Dateien nur angezeigt werden können.

```
WINDOWP crt_edit (OBJECT *obj, OBJECT *menu, WORD icon,
                  BYTE *filename);
```

Ein Fenster zur Darstellung einer ASCII-Datei wird kreiert.

| | |
|---|---|
| crt_edit | : Zeiger auf das kreierte Fenster oder NULL, wenn es nicht kreiert werden konnte |
| obj | : Zeiger auf den Objektbaum, welcher sich im Fenster befinden soll |
| menu | : Menüzeile, welche sich im Fenster befinden soll, oder NULL, wenn es keine Menüzeile geben soll |
| icon | : Index des Piktogramms, aus dem das Fenster entstehen soll, oder NIL, wenn es kein solches Piktogramm gibt |
| filename | : Zeiger auf den Namen der ASCII-Datei |

Ist kein Pfad im Dateinamen angegeben, so wird der aktuelle Pfad mit aufgenommen.

BOOLEAN open_edit (WORD icon, BYTE *filename);

Ein Fenster zur Darstellung einer ASCII-Datei wird geöffnet.

| | |
|---|---|
| open_edit | : TRUE, wenn das Fenster geöffnet werden konnte, FALSE sonst |
| icon | : Index des Piktogramms, aus dem das Fenster entstehen soll, oder NIL, wenn es kein solches Piktogramm gibt |
| filename | : Zeiger auf den Namen der ASCII-Datei |

Wenn ein ganz bestimmtes Piktogramm, z.B. aus dem Klemmbrettfenster, gemeint ist, so wird dieses nach oben gebracht, wenn es bereits offen ist. Ansonsten wird ein neues Fenster kreiert, wenn nicht bereits ein geschlossenes Fenster vorhanden ist. Dieses Fenster wird dann geöffnet.

BOOLEAN info_edit (WINDOWP window, WORD icon);

Die Information der ASCII-Datei wird angezeigt.

info_edit : TRUE, wenn es Informationen gibt, FALSE sonst
window : Zeiger auf das Fenster, von dem Informationen gezeigt werden sollen
icon : Index des Piktogramms, von dem Informationen gezeigt werden sollen

BOOLEAN help_edit (WINDOWP window, WORD icon);

Eine Hilfemeldung für die ASCII-Datei wird angezeigt.

help_edit : TRUE, wenn es eine Hilfemeldung gibt, FALSE sonst
window : Zeiger auf das Fenster, von dem eine Hilfemeldung gezeigt werden soll
icon : Index des Piktogramms, von dem eine Hilfemeldung gezeigt werden soll

Für die Parameter gilt das bei „help_desktop" Gesagte.

BOOLEAN init_edit (VOID);

Das Modul wird initialisiert.

init_edit : TRUE, wenn die Initialisierung geklappt hat, FALSE sonst

BOOLEAN term_edit (VOID);

Das Modul wird terminiert.

term_edit : TRUE, wenn die Terminierung geklappt hat, FALSE sonst

### 6.5.17 Modul POWER

Das Modul dient der Darstellung von Potenzen einer Zahl in einem Fenster. Dazu kommt ein Beispiel für die Benutzung eines Pop-Up-Menüs.

WINDOWP crt_power (OBJECT *obj, OBJECT *menu, WORD icon);

Ein Fenster zur Darstellung von Potenzen einer Zahl wird kreiert.

crt_power : Zeiger auf das kreierte Fenster oder NULL, wenn es nicht kreiert werden konnte
obj : Zeiger auf den Objektbaum, welcher sich im Fenster befinden soll
menu : Menüzeile, welche sich im Fenster befinden soll, oder NULL, wenn es keine Menüzeile geben soll
icon : Index des Piktogramms, aus dem das Fenster entstehen soll, oder NIL, wenn es kein solches Piktogramm gibt

Das kreierte Fenster wird eine eigene Mausform haben (USER_DEF).

Beim Kreieren werden unter anderem auch die Funktionen der Fensterstruktur eingehängt. Dadurch kann der Window-Manager diese Funktionen bei Bedarf aufrufen. Eine dieser Funktion ist die „click"-Funktion, die aufgerufen wird, wenn in das Fenster geklickt wird. Sie ist lokal in diesem Modul als „wi_click" definiert.

Nach dem Klick wird zunächst ein etwaiges anderes Fenster deselektiert. Dann werden im Menübaum „popup" bei allen Menüs des Objekts POWERS die Häkchen entfernt. Die aktuell eingestellte Potenz wird abgehakt. Dann wird die Funktion „popup_menu" aufgerufen. An sie wird das Pop-Up-Menü übergeben. Das Menü soll relativ zur Mausposition ohne zusätzlichen Offset erscheinen. Außerdem soll die Mausform genau in der Mitte der aktuell eingestellten Potenz erscheinen (zentriert). Es wird dann auf das Klicken oder Loslassen des linken Mausknopfes gewartet.

BOOLEAN open_power (WORD icon);

Ein Fenster zur Darstellung von Potenzen einer Zahl wird geöffnet.

open_power : TRUE, wenn das Fenster geöffnet werden konnte, FALSE sonst
icon : Index des Piktogramms, aus dem das Fenster entstehen soll, oder NIL, wenn es kein solches Piktogramm gibt

Es wird jeweils gesucht, ob noch ein geschlossenes Fenster dieser Klasse vorhanden ist. Wenn ja, dann wird dieses geöffnet, ansonsten wird ein neues Fenster kreiert und dieses geöffnet.

BOOLEAN info_power (WINDOWP window, WORD icon);

Die Information des Fensters wird angezeigt.

info_power : TRUE, wenn es Informationen gibt, FALSE sonst
window : Zeiger auf das Fenster, von dem Informationen gezeigt werden sollen
icon : Index des Piktogramms, von dem Informationen gezeigt werden sollen

BOOLEAN help_power (WINDOWP window, WORD icon);

Eine Hilfemeldung für das Potenzfenster wird angezeigt.

help_power: TRUE, wenn es eine Hilfemeldung gibt, FALSE sonst
window : Zeiger auf das Fenster, von dem eine Hilfemeldung gezeigt werden soll
icon : Index des Piktogramms, von dem eine Hilfemeldung gezeigt werden soll

Für die Parameter gilt das bei „help_desktop" Gesagte.

BOOLEAN init_power (VOID);

Das Modul wird initialisiert.

init_power : TRUE, wenn die Initialisierung geklappt hat, FALSE sonst

BOOLEAN term_power (VOID);

Das Modul wird terminiert.

term_power: TRUE, wenn die Terminierung geklappt hat, FALSE sonst

### 6.5.18 Modul GRAF

Das Modul dient der Darstellung einer Grafik (Ellipse) in verschiedenen Formen. Außerdem wird ein Beispiel für eine Art Multitasking gegeben.

WINDOWP crt_graf (OBJECT *obj, OBJECT *menu, WORD icon);

Ein Fenster zur Darstellung von Grafik wird kreiert.

crt_graf : Zeiger auf das kreierte Fenster oder NULL, wenn es nicht kreiert werden konnte
obj : Zeiger auf den Objektbaum, welcher sich im Fenster befinden soll
menu : Menüzeile, welche sich im Fenster befinden soll, oder NULL, wenn es keine Menüzeile geben soll
icon : Index des Piktogramms, aus dem das Fenster entstehen soll, oder NIL, wenn es kein solches Piktogramm gibt

Beim Kreieren werden unter anderem auch die Funktionen der Fensterstruktur eingehängt. Dadurch kann der Window-Manager diese Funktionen bei Bedarf aufrufen. Eine dieser Funktionen ist die „timer"-Funktion, die aufgerufen wird, wenn eine bestimmte Zeitspanne vergangen ist. Sie ist lokal in diesem Modul als „wi_timer" definiert.

Am Anfang des Moduls wird die Zeit als Macro definiert:

```
#define MILLI 1000
```

Das bedeutet, daß jeweils nach Ablauf einer Sekunde das Fenster eine Aktion durchführen kann, wenn eine Funktion dafür vorhanden und in die Fensterstruktur eingeklinkt ist. In unserer Funktion „wi_timer" wird ein Tastaturereignis für die Taste '+' für dieses Fenster simuliert. Diese Taste kann auch unabhängig vom Zeitereignis gedrückt werden. Dadurch wird das Muster, welches im Fenster in einer Ellipse gezeichnet wird, weitergeschaltet. Es handelt sich dabei um die GEM-Muster aus Abb. 2.8.

Das Zeitereignis kann immer dann auftreten, wenn in der Hauptereignisschleife (Modul EVENT) gewartet wird. Das bedeutet, daß Dialogboxen diese Ereignisse bremsen. Die Tätigkeit, die ein Fenster nach einem Zeitereignis durchführt, kann mannigfaltig sein. So könnte sich hinter diesem Fenster ein Druckerspooler verbergen, der im Hintergrund Daten an den Drucker sendet. Ein Fenster, das Text anzeigt, könnte nach Ablauf einer bestimmten Zeit diesen Text abspeichern usw.

```
BOOLEAN open_graf (WORD icon);
```

Ein Fenster zur Darstellung von Grafik wird geöffnet.

| | | |
|---|---|---|
| open_graf | : | TRUE, wenn das Fenster geöffnet werden konnte, FALSE sonst |
| icon | : | Index des Piktogramms, aus dem das Fenster entstehen soll, oder NIL, wenn es kein solches Piktogramm gibt |

Es wird jeweils gesucht, ob noch ein geschlossenes Fenster dieser Klasse vorhanden ist. Wenn ja, dann wird dieses geöffnet, ansonsten wird ein neues Fenster kreiert und dieses geöffnet.

```
BOOLEAN info_graf (WINDOWP window, WORD icon);
```

Die Information des Fensters wird angezeigt.

| | | |
|---|---|---|
| info_graf | : | TRUE, wenn es Informationen gibt, FALSE sonst |
| window | : | Zeiger auf das Fenster, von dem Informationen gezeigt werden sollen |
| icon | : | Index des Piktogramms, von dem Informationen gezeigt werden sollen |

BOOLEAN help_graf (WINDOWP window, WORD icon);

Eine Hilfemeldung für das Grafikfenster wird angezeigt.

help_graf : TRUE, wenn es eine Hilfemeldung gibt, FALSE sonst
window : Zeiger auf das Fenster, von dem eine Hilfemeldung gezeigt werden soll
icon : Index des Piktogramms, von dem eine Hilfemeldung gezeigt werden soll

Für die Parameter gilt das bei „help_desktop" Gesagte.

BOOLEAN init_graf (VOID);

Das Modul wird initialisiert.

init_graf : TRUE, wenn die Initialisierung geklappt hat, FALSE sonst

BOOLEAN term_graf (VOID);

Das Modul wird terminiert.

term_graf : TRUE, wenn die Terminierung geklappt hat, FALSE sonst

### 6.5.19 Modul MODULE

Dieses Modul hat noch keine Aufgabe. Es dient als Grundgerüst für das Schreiben von eigenen Modulen. Es ist so aufgebaut, daß alle Funktionen, die ein Modul benötigt, welches Fenster benutzt, vorhanden sind. Aus diesem Grund soll das Modul eingehender besprochen werden. Hier werden auch alle diejenigen Funktionen vorgestellt, die nur lokal zu dem Modul definiert sind und bei der Besprechung der anderen Module zu kurz gekommen sind.

Die Schnittstelle des Moduls MODULE entspricht der Schnittstelle des Moduls CLIPBRD. Anstatt von „open_clipbrd" taucht hier aber z.B. ein „open_module" auf. Das bedeutet, daß im Quelltext alle Vorkommen von „module" durch den Namen des Moduls ersetzt werden sollen, also z.B. durch „draw", wenn es sich um ein Zeichenmodul handelt. Das gleiche gilt für alle Vorkommen des Wortes „MODULE", welches dann analog zu oben durch „DRAW" ersetzt werden sollte.

Außerdem muß der Konstanten „CLASS_MODULE", die in unserem Beispiel durch „CLASS_DRAW" ersetzt werden würde, ein Wert zugeordnet werden, der noch nicht eine andere Klasse beschreibt.

a) Globale Funktionen

BOOLEAN icons_module (WORD src_obj, WORD dest_obj);

Operationen für das Piktogramm des Moduls werden durchgeführt. Diese Funktion wird vom übergeordneten Modul aufgerufen, wenn das Piktogramm des Moduls auf ein anderes Piktogramm verschoben wird. Die Funktion muß nicht existieren, wenn es kein Piktogramm gibt, welches das Modul repräsentiert.

icons_module: TRUE, wenn es eine gültige Piktogramm-Operation gab, FALSE sonst
src_obj : Index des Quellobjektes
dest_obj : Index des Zielobjektes

Falls es ein Fenster gibt, welches zu einem Piktogramm gehört, so wird dieses zunächst gesucht. In der zugehörigen Fensterstruktur sind mit großer Wahrscheinlichkeit Informationen zu finden, die benutzt werden können, um die Operation auszuführen (siehe auch „icons_clipbrd").

WINDOWP crt_module (OBJECT *obj, OBJECT *menu, WORD icon);

Ein Fenster des Moduls wird kreiert.

crt_module : Zeiger auf das kreierte Fenster oder NULL, wenn es nicht kreiert werden konnte
obj : Zeiger auf den Objektbaum, welcher sich im Fenster befinden soll
menu : Menüzeile, welche sich im Fenster befinden soll, oder NULL, wenn es keine Menüzeile geben soll
icon : Index des Piktogramms, aus dem das Fenster entstehen soll, oder NIL, wenn es kein solches Piktogramm gibt

Beim Kreieren werden die schon existierenden Fenster der gleichen Klasse gezählt (inx). Neue Fenster dieser Klasse öffnen sich dann leicht versetzt, so daß sie sich nicht vollständig verdecken.

Anschließend wird ein Fenster über den Aufruf „create_window" kreiert.

Wenn das Kreieren geklappt hat, so wird die Höhe der Menüzeile berechnet, falls das Fenster eine Menüzeile besitzt. Nun werden die einzelnen Komponenten der Fensterstruktur gefüllt. Für die Flags wird das entsprechende Macro benutzt, das am Anfang des Moduls definiert wurde.

Die Dokumentposition wird auf den linken oberen Punkt gesetzt, die Dokumentbreite auf die anfänglich gesetzte Breite und die Dokumenthöhe auf null. Die beiden letzten Werte hängen natürlich vom Inhalt des Dokuments ab. Bei einem Texteditor würden hier die Anzahl der Zeilen und die maximale Anzahl der Spalten eingetragen werden.

Der X-Faktor und der Y-Faktor werden durch Makros gesetzt, die im Modul definiert wurden. Das gleiche gilt für den Scrollbereich. Bei dessen Initialisierung wird zusätzlich noch der am Anfang berechnete Index benutzt, damit die Bereiche nicht übereinander zu liegen kommen. Bei der Y-Position muß außerdem darauf geachtet werden, daß sie immer an einer geraden Position beginnt. Das Makro „INITY" ist bereits gerade, die Variable „gl_hbox" ebenfalls. Nun hängt es nur noch von der Höhe der Menüzeile ab, ob der Wert gerade ist oder nicht. Ist die Höhe ungerade, so wird ein Pixel addiert.

Der Workbereich wird nun ebenfalls gesetzt. Er ist so groß wie der Scrollbereich zuzüglich der eventuell vorhandenen Menüzeile, für die das Fenster ja nicht verantwortlich ist. Will man einen größeren Workbereich haben, weil man noch Randkomponenten benötigt (Panels etc. siehe Abb. 6.3), so muß man entsprechende Werte vom linken bzw. vom oberen Rand abziehen und in der Breite bzw. der Höhe hinzuaddieren (siehe auch Abb. 6.4). Im Modul CLIPBRD wird dies über zwei Makros bewerkstelligt.

Die Mausnummer wird nun eingetragen. Falls sie den Wert 255 (= USER_DEF) erhält, so muß auch die Adresse einer Mausform eingetragen werden (siehe auch Modul POWER).

Die Anzahl der Millisekunden, die vergehen sollen, bevor das Fenster eine Aktion durchführen darf, wird über ein Makro eingetragen. In unserem Beispiel sind dies 1000 Millisekunden, also eine Sekunde. Wir erinnern uns, daß die Auflösung im „evnt multi"-Aufruf des Moduls EVENT auf 100 Millisekunden (1/10 Sekunde) eingestellt ist. Falls man kleinere Werte benötigt, so muß man die Konstante im Modul EVENT verkleinern. Wir weisen jedoch nochmals ausdrücklich darauf hin, daß zu klein gewählte Werte nicht genau sind. So kann man bei Einstellen des Wertes 1 nicht erwarten, daß in jeder Millisekunde ein Ereignis ausgelöst wird. Dies hängt vom Timer des Rechners und der Genauigkeit des GEM ab. Da dies aber von Rechner zu Rechner verschieden ist, sollte man auf zu kleine Werte verzichten.

Der Wert „special" der Fensterstruktur wird ebenfalls eingetragen. Meistens handelt es sich um einen Zeiger auf irgendeine Struktur, die dann beliebige Informationen beinhalten kann, die zu einem Fenster gehören (siehe z.B. Modul EDIT).

Der Objektbaum und die Menüzeile werden aus den Parametern von „crt_module" eingetragen.

Nun folgt das Eintragen aller Funktionen, die unter anderem vom Window-Manager aufgerufen werden können. Alle diese Routinen sind lokal in diesem Modul definiert und beginnen mit dem Präfix „wi_". Dies gilt für alle Module. Da sie lokal definiert sind, werden die Bezeichner nicht nach außen getragen, so daß es zu keinen Konflikten kommen kann. Die Funktionen werden weiter unten erklärt. Möchte man an eine Stelle keine Funktion eintragen, weil eine bestimmte Operation nicht ausführbar ist (z.B. Anklicken des Fensterinnern), so trägt man den Wert NULL ein. Dies ist allerdings nicht unbedingt nötig, da beim Kreieren eines Fensters alle Komponenten mit null gefüllt werden.

Am Ende werden noch die Komponenten „name" und „info" der Fensterstruktur gesetzt. Wir haben für die beiden Zeichenketten jeweils Texte in der Resource-Datei definiert. Dies hängt natürlich von der jeweiligen Anwendung ab. Beispiele sind in allen Modulen zu finden, die Fenster öffnen.

BOOLEAN open_module (WORD icon);

Ein Fenster des Moduls wird geöffnet.

open_module : TRUE, wenn das Fenster geöffnet werden konnte, FALSE sonst
icon : Index des Piktogramms, aus dem das Fenster entstehen soll, oder NIL, wenn es kein solches Piktogramm gibt

In diesem Beispiel gehen wir davon aus, daß es im Prinzip beliebig viele Fenster (aber maximal sieben wegen GEM) geben kann. Zunächst wird gesucht, ob es bereits ein geschlossenes Fenster gibt, das der gleichen Klasse angehört und aus demselben Piktogramm entstanden ist (oder gar keinem). Dieses wird dann geöffnet. Gibt es kein solches Fenster, so wird ein neues kreiert. Im Erfolgsfall wird das Fenster geöffnet.

BOOLEAN info_module (WINDOWP window, WORD icon);

Die Information des Fensters oder des Piktogramms, aus dem das Fenster entstanden ist, wird angezeigt.

info_module : TRUE, wenn es Informationen gibt, FALSE sonst
window : Zeiger auf das Fenster, von dem Informationen gezeigt werden sollen
icon : Index des Piktogramms, von dem Informationen gezeigt werden sollen

Zunächst wird der Parameter „icon" ausgewertet. Ist er ungleich NIL, so handelt es sich um ein ganz bestimmtes Fenster, das aus einem ganz bestimmten Piktogramm erzeugt wurde. Dieses Fenster wird nun über „search_window" gesucht.

Wurde ein solches Fenster gefunden oder war kein Piktogramm angegeben (NIL), dafür aber ein Fenster, so wird eine entsprechende Notiz ausgegeben. Es kann aber auch eine zu diesem Modul lokale Funktion aufgerufen werden, die je nach Fenster oder Piktogramm unterschiedliche Informationen ausgeben kann (siehe auch Modul EDIT).

BOOLEAN help_module (WINDOWP window, WORD icon);

Eine Hilfemeldung für das Fenster oder Piktogramm wird angezeigt.

help_module: TRUE, wenn es eine Hilfemeldung gibt, FALSE sonst
window : Zeiger auf das Fenster, von dem eine Hilfemeldung gezeigt werden soll
icon : Index des Piktogramms, von dem eine Hilfemeldung gezeigt werden soll

Die Parameter „window" und „icon" sind hier ohne Bedeutung und nur wegen der Vollständigkeit der Schnittstelle mit aufgenommen worden. Sie werden genau dann benötigt, wenn man zu verschiedenen Inkarnationen derselben Fensterklasse verschiedene Hilfemeldungen benötigt. Dann zeigt „window" nämlich auf das entsprechende Fenster. Ist dieses nicht bekannt, aber das Piktogramm, aus dem das Fenster entstanden ist, so kann man auch dieses angeben. Über ein „search_window" (ähnlich wie bei „info_module") kann man dann das eindeutig zugehörige Fenster finden.

BOOLEAN init_module (VOID);

Das Modul wird initialisiert.

init_module: TRUE, wenn die Initialisierung geklappt hat, FALSE sonst

In dieser Funktion können Variablen initialisiert werden, Speicherplatz angefordert werden usw. Beispiele finden sich unter anderem in den Modulen GLOBAL und WINDOWS.

BOOLEAN term_module (VOID);

Das Modul wird terminiert.

term_clipbrd: TRUE, wenn die Terminierung geklappt hat, FALSE sonst

Falls beim Initialisieren des Moduls Speicher angefordert wurde, so wird dieser hier wieder freigegeben.

b) Lokale Definitionen

Im Modul werden eine Vielzahl von lokalen Definitionen zu finden sein, die nur das jeweilige Modul kennt. Nach dem Einbinden der verschiedenen Headerdateien werden zunächst Makros definiert.

KIND gibt die Art des Fensters, d.h. die GEM-Komponenten an.

FLAGS gibt die Flags aus WINDOWS.H an, die gesetzt werden können.

XFAC ist der X-Faktor, der beim horizontalen Scrolling berücksichtigt wird

YFAC ist der Y-Faktor, der beim vertikalen Scrolling berücksichtigt wird

INITX, INITY, INITW und INITH definieren das Rechteck des Scrollbereichs. Dieses wird immer angegeben. Der Workbereich wird dann erst in „crt_module" errechnet.

MILLI gibt die Anzahl der Millisekunden an, auf die ein Fenster warten möchte, bevor es eine Aktion ausführt.

c) Lokale Funktionen

Vor der Implementierung der lokalen Funktionen werden zunächst alle Prototypen angegeben. Alle Funktionen, die in die Fensterstruktur eingehängt werden, wurden schon in Kapitel 6.3 bei der Beschreibung der Struktur gegeben. Aus diesem Grund sollen sie hier nur noch kurz erwähnt werden.

VOID update_menu (WINDOWP window);

Die Funktion wird in die Komponente „updt_menu" der Fensterstruktur eingehängt, wenn eine Menüzeile im Fenster dargestellt werden soll.

VOID handle_menu (WINDOWP window, WORD title, WORD item);

Die Funktion wird in die Komponente „hndl_menu" der Fensterstruktur eingehängt, wenn eine Menüzeile im Fenster dargestellt werden soll.

VOID box (WINDOWP window, BOOLEAN grow);

Die Funktion zeichnet eine sich ausdehnende oder zusammenziehende Box. Sie wird von den Funktionen „wi_open" und „wi_close" aufgerufen.

| | | |
|---|---|---|
| window | : | Zeiger auf das Fenster, an welchem die Box enden soll bzw. von welchem die Box ausgehen soll |
| grow | : | TRUE, wenn eine sich vergrößernde Box gezeichnet werden soll, FALSE, wenn eine sich zusammenziehende Box gezeichnet werden soll |

Die Funktion besorgt sich zunächst das kleine Rechteck, welches aus einem Icon oder einem Menü kommen kann. Dann wird der Rand des Fensters berechnet und schließlich die Growbox oder Shrinkbox gezeichnet. Falls es sich um ein Accessory handelt, wird keine Shrinkbox gezeichnet, da die Meldung AC_CLOSE an einer strategisch ungünstigen Stelle an das Accessory ergeht. Beim Schließen der Accessory-Fenster würden dann Shrinkboxen auch im „weißen" Bildschirm erscheinen.

BOOLEAN wi_test (WINDOWP window, WORD action);

Die Funktion wird in die Komponente „test" der Fensterstruktur eingehängt. Sie wird vom Window-Manager aufgerufen, wenn ein Fenster geschlossen oder gelöscht werden

soll (DO_CLOSE bzw. DO_DELETE). Ansonsten wird die Funktion vom globalen Menü-Manager aufgerufen, wenn getestet werden soll, ob die Cut/Copy/Paste-Menüpunkte ausführbar sind oder nicht. Ist dies der Fall, so muß die Funktion TRUE zurückliefern, FALSE sonst.

Wird die Funktion nicht eingehängt, so geht man davon aus, daß es keine Cut/Copy/Paste-Funktionalität gibt.

VOID wi_open (WINDOWP window);

Die Funktion wird in die Komponente „open" der Fensterstruktur eingehängt. Sie wird vom Window-Manager aufgerufen, bevor das Fenster geöffnet wird. An dieser Stelle kann z.B. ein sich ausdehnendes Rechteck gezeichnet werden.

VOID wi_close (WINDOWP window);

Die Funktion wird in die Komponente „close" der Fensterstruktur eingehängt. Sie wird vom Window-Manager aufgerufen, nachdem das Fenster geschlossen wurde. An dieser Stelle kann z.B. ein sich zusammenziehendes Rechteck gezeichnet werden.

VOID wi_delete (WINDOWP window);

Die Funktion wird in die Komponente „delete" der Fensterstruktur eingehängt. Sie wird vom Window-Manager aufgerufen, bevor das Fenster gelöscht wird. An dieser Stelle kann z.B. der Fensterinhalt gerettet werden.

VOID wi_draw (WINDOWP window);

Die Funktion wird in die Komponente „draw" der Fensterstruktur eingehängt. Sie wird vom Window-Manager aufgerufen, wenn der Fensterinhalt gezeichnet wird. Das aktuelle Clipping-Rechteck ist bereits gesetzt.

VOID wi_arrow (WINDOWP window, WORD dir, LONG oldpos, LONG newpos);

Die Funktion wird in die Komponente „arrow" der Fensterstruktur eingehängt. Sie wird vom Window-Manager aufgerufen, wenn der Fensterinhalt gescrollt werden soll.

Zunächst hat das Fenster die Möglichkeit, auf die Parameter zu reagieren. So kann es z.B. weitere Textzeilen eines Dokumentes anfordern oder im Datenbestand vorrücken. Dann werden die neue Position im Dokument festgelegt, die Schieber gesetzt und das eigentliche Scrolling durchgeführt.

VOID wi_snap (WINDOWP window, RECT *new, WORD mode);

Die Funktion wird in die Komponente „snap" der Fensterstruktur eingehängt. Sie wird vom Window-Manager aufgerufen, wenn das Fenster bewegt oder in seiner Größe verändert wurde.

Zunächst wird die Differenz zur vorigen Position bestimmt. Dann wird entschieden, welche Komponenten des Fensters einrasten sollen (X-, Y-Position, Breite, Höhe). Beim Vergrößern oder Verkleinern können außerdem spezielle Algorithmen ausgeführt werden. So ist es günstig, beim Vergrößern eines Fensters unter den unteren Rand eines Dokumentes neue Information von weiter oben zu holen und das untere Ende exakt darzustellen, d.h. das untere Ende des Fensters mit dem unteren Ende des Dokumentes abzugleichen.

VOID wi_objop (WINDOWP window, SET objs, WORD action);

Die Funktion wird in die Komponente „objop" der Fensterstruktur eingehängt. Sie wird vom Menü-Manager aufgerufen, wenn die Funktionen „Öffnen", „Info" und „Hilfe" gewählt werden, wobei gleichzeitig ein oder mehrere Objekte selektiert waren.

WORD wi_drag (WINDOWP src_window, WORD src_obj,
WINDOWP dest_window, WORD dest_obj);

Die Funktion wird in die Komponente „drag" der Fensterstruktur eingehängt. Sie wird vom Window-Manager aufgerufen, wenn die Applikation bekanntgibt, daß ein Objekt von einem Fenster in ein anderes geschoben wurde.

VOID wi_click (WINDOWP window, MKINFO *mk);

Die Funktion wird in die Komponente „click" der Fensterstruktur eingehängt. Sie wird vom Window-Manager aufgerufen, wenn in das Fenster geklickt wurde.

VOID wi_unclick (WINDOWP window);

Die Funktion wird in die Komponente „unclick" der Fensterstruktur eingehängt. Sie wird vom Window-Manager aufgerufen, um ein Fenster zu deselektieren, wenn in ein anderes geklickt wurde.

BOOLEAN wi_key (WINDOWP window, MKINFO *mk);

Die Funktion wird in die Komponente „key" der Fensterstruktur eingehängt. Sie wird vom Window-Manager aufgerufen, wenn eine Taste gedrückt wurde. Handelte es sich um eine Taste für das Menü, welches in das Fenster eingehängt werden kann, so muß nach dem Ausführen des Menüs die Funktion sofort beendet werden.

VOID wi_timer (WINDOWP window);

Die Funktion wird in die Komponente „timer" der Fensterstruktur eingehängt. Sie wird vom Window-Manager aufgerufen, wenn eine bestimmte Zeit für ein Fenster abgelaufen ist. Das Fenster darf dann eine Aktion durchführen.

VOID wi_top (WINDOWP window);

Die Funktion wird in die Komponente „top" der Fensterstruktur eingehängt. Sie wird vom Window-Manager aufgerufen, wenn das Fenster nach oben kommt.

VOID wi_untop (WINDOWP window);

Die Funktion wird in die Komponente „untop" der Fensterstruktur eingehängt. Sie wird vom Window-Manager aufgerufen, wenn das Fenster nach unten kommt.

VOID wi_edit (WINDOWP window, WORD action);

Die Funktion wird in die Komponente „edit" der Fensterstruktur eingehängt. Sie wird vom Menü-Manager aufgerufen, wenn einer der Menüpunkte Undo, Cut, Copy, Paste, Clear oder Select all aufgerufen wird.

## 6.6 Bedienung von SCRAP

Wenn alle Module übersetzt und gelinkt wurden, so ensteht ein Programm mit Namen SCRAP.PRG (GEMDOS), SCRAP.APP (MS-DOS) oder SCRAP.286 (FlexOS). Dieses kann durch Doppelklick vom Desktop aus gestartet werden.

Zunächst wird der Desktop dargestellt, dann öffnet sich auch das Klemmbrettfenster. Falls sich Dateien im Verzeichnis des GEM-Klemmbretts befinden, werden diese als Piktogramme angezeigt. Falls sich keine Dateien dort befinden, sollten Sie wenigstens eine Text-Datei dort hineinkopieren, damit Sie mit dem Programm richtig arbeiten können. Nennen Sie die Datei z.B. SCRAP.TXT.

Die Dateien im Klemmbrettfenster können geöffnet und damit angezeigt werden. Dies geschieht in der gewohnten Weise, d.h. durch Doppelklick oder Einfachklick und nachfolgendem Öffnen über den entsprechenden Menüpunkt. Auf die gleiche Weise können auch mehrere Objekte selektiert und geöffnet werden. Für die Objekte im Desktop gilt dies analog.

Bestimmte einzelne Menüpunkte können wahlweise auch über die Funktionstasten angewählt werden. Im Klemmbrettfenster befindet sich auch eine Menüzeile, die angeklickt werden kann. Dann klappt das jeweilige Menü herunter. Aus diesem können nun ebenfalls Menüpunkte ausgewählt werden. Dazu muß der Mausknopf entweder gedrückt oder losgelassen werden, je nachdem, ob das Menü mit gedrückt gehaltenem Mauskopf angewählt wurde oder nicht. Das Menü „Anzeigen" im Klemmbrett bietet die Möglichkeit, aus verschiedenen Masken auszuwählen, so daß nur ein Teil der Dateien im Klemmbrett angezeigt werden kann.

Die Menüpunkte „Ausschneiden", „Kopieren" und „Einfügen" des Menüs „Bearbeiten" können nur dann auf die Objekte des Klemmbrettfensters angewendet werden, wenn der Schalter „Auf GEM-Klemmbrett" eingeschaltet ist. Sie haben dann folgende Bedeutung:

Ausschneiden: Das Original wird gelöscht, aber eine Kopie davon auf das Klemmbrett abgelegt. Dies entspricht einem Umbenennen. Heißt das Original „TEST.TXT", so wird also zunächst ein eventuell vorhandenes „SCRAP.TXT" gelöscht, dann „TEST.TXT" in „SCRAP.TXT" umbenannt. Dies bedeutet, daß danach immer eine Datei mit demselben Suffix wie das Original existiert, jedoch das Präfix SCRAP hat.

Kopieren: Wie Ausschneiden, wobei das Original erhalten bleibt. Es existieren also dann zwei identische Dateien, z.B. „TEST.TXT" und „SCRAP.TXT".

Einfügen: Hier nicht implementiert.

Der Menüpunkt „Löschen" ist aktiv, wenn mindestens ein Objekt im Klemmbrettfenster angewählt ist. „Alles auswählen" ist immer aktiv, wenn mindestens ein Objekt im Klemmbrettfenster existiert.

Die beiden Menüpunkte „Power" und „Grafik" öffnen jeweils bis zu sieben Fenster der entsprechenden Sorte. Die Fenster „Power" zeigen die Potenzen einer Zahl. Auf eine neue Potenz kann umgeschaltet werden, wenn eine Zahl zwischen zwei und neun eingegeben wird, während das Fenster an oberster Stelle liegt. Die zweite Möglichkeit wird durch ein Pop-Up-Menü gegeben, welches an der Stelle des Mauszeigers erscheint, wenn in das Fenster geklickt wird. Dann kann aus diesem die Potenz ausgewählt werden. Das Fenster demonstriert außerdem eine eigene Mausform, die beim Eintreten in das Fenster erscheint.

Die Grafikfenster demonstrieren den Gebrauch verschiedener Füllmuster. Die Füllmuster können durch die Tasten '+' und '-' vor- bzw. zurückgeschaltet werden. Mit den Tasten '0' bis '3' können die verschiedenen Füllarten angezeigt werden (siehe auch Abb. 2.8). Außerdem führt das Fenster die Benutzung des Timers vor. Jede Sekunde wird auf ein

neues Füllmuster umgeschaltet. Wurden mehrere Fenster geöffnet und verschiedene Füllmuster bzw. Füllarten ausgewählt, so ergibt sich der Effekt eines Multitasking-Systems. Die Benutzung des Timers kann für jedes Fenster durch die Taste '*' unterbrochen und wieder fortgesetzt werden.

Der Menüpunkt „Einstellungen" im Menü „Optionen" demonstriert sowohl benutzerdefinierte Objekte (ankreuzbare Knöpfe) als auch die Implementierung einer Hilfe-Box.

Das Programm kann auch als Accessory gestartet werden (nur Atari ST), wenn es in SCRAP.ACC umbenannt und auf das Bootlaufwerk kopiert wird. Beim Öffnen des Accessories wird nun zunächst der Desktop in einem echten Fenster gezeigt. In ihm befindet sich auch die globale Menüzeile. Zum Öffnen des Klemmbretts muß nun noch ein Doppelklick auf das Klemmbrettsymbol getätigt werden. Ansonsten wird das Accessory wie die Applikation bedient.

Das Demoprogramm enthält natürlich viele Dinge, die nur der Demonstration dienen. Wenn auch Sie das Gefühl haben, daß es intuitiv zu bedienen ist, so nehmen Sie sich an ihm ein Beispiel und programmieren Ihre Applikationen mit Hilfe unserer Routinen. Allein schon die ähnliche Menügestaltung wird dem Benutzer eine große Hilfe sein, so daß er nicht immer wieder umdenken muß.

# Anhang A. Header-Dateien

## Header-Datei PORTAB.H:

```
/*************************************************************************/
/*                                                                       */
/* PORTAB.H                                                              */
/*                                                                       */
/* Use of this file may make your code compatible with all C compilers   */
/* listed.                                                               */
/*                                                                       */
/*************************************************************************/

/*************************************************************************/
/* ENVIRONMENT                                                           */
/*************************************************************************/

#ifndef __PORTAB__
#define __PORTAB__

#define GEMDOS     1                    /* Digital Research GEMDOS     */
#define MSDOS      0                    /* Microsoft MSDOS             */
#define OS2        0                    /* Microsoft OS/2              */
#define FLEXOS     0                    /* Digital Research FlexOS     */

#define M68000     1                    /* Motorola Processing Unit    */
#define I8086      0                    /* Intel Processing Unit       */

#define DR_C       0                    /* Digital Research C Compiler */
#define LASER_C    0                    /* Laser C Compiler            */
#define LATTICE_C  0                    /* Lattice C Compiler          */
#define MW_C       0                    /* Mark Williams C Compiler    */
#define TURBO_C    1                    /* Turbo C Compiler            */
#define MS_C       0                    /* Microsoft C Compiler        */
#define HIGH_C     0                    /* Metaware High C Compiler    */
```

```
#define GEM1        0x0001                  /* ATARI GEM version          *
#define GEM2        0x0002                  /* MSDOS GEM 2.x versions     *
#define GEM3        0x0004                  /* MSDOS GEM/3 version        *
#define XGEM        0x0100                  /* OS/2,FlexOS X/GEM version  *

#ifndef GEM
#if GEMDOS
#define GEM         GEM1                    /* GEMDOS default is GEM1     *
#endif /* GEMDOS */

#if MSDOS
#define GEM         GEM3                    /* MSDOS default is GEM3      *
#endif /* MSDOS */

#if OS2
#define GEM         XGEM                    /* OS/2 default is X/GEM      *
#endif /* MSDOS */

#if FLEXOS
#define GEM         XGEM                    /* FlexOS default is X/GEM    *
#endif /* FLEXOS */
#endif /* GEM */

/*****************************************************************************
/* STANDARD TYPE DEFINITIONS                                                 *
/*****************************************************************************

#define BYTE    char                        /* Signed byte                *
#define UBYTE   unsigned char               /* Unsigned byte              *

#if LATTICE_C
#define WORD    short                       /* Signed word (16 bits)      *
#define UWORD   unsigned short              /* unsigned word              *
#else
#define WORD    int                         /* Signed word (16 bits)      *
#define UWORD   unsigned int                /* unsigned word              *
#endif

#define LONG    long                        /* signed long (32 bits)      *
#define ULONG   unsigned long               /* Unsigned long              *

#define BOOLEAN WORD                        /* 2 valued (true/false)      *

#define FLOAT   float                       /* single precision float     *
#define DOUBLE  double                      /* double precision float     *
```

```
#define INT     int                             /* a machine dependent int     */
#define UINT    unsigned int                    /* a machine dependent uint    */

#define REG     register                        /* register variable           */
#define AUTO    auto                            /* Local to function           */
#define EXTERN  extern                          /* External variable           */
#define LOCAL   static                          /* Local to module             */
#define MLOCAL  LOCAL                           /* Local to module             */
#define GLOBAL                                  /* Global variable             */

/*****************************************************************************/
/* COMPILER DEPENDENT DEFINITIONS                                            */
/*****************************************************************************/

#if GEMDOS                                      /* GEMDOS compilers            */
#if DR_C
#define void WORD                               /* DR_C doesn't know void      */
#endif /* DR_C */

#if LASER_C
#define graf_mbox graf_movebox                  /* Wrong GEM binding           */
#define graf_rubbox graf_rubberbox              /* Wrong GEM binding           */
#endif /* LASER_C */

#if LATTICE_C
#define graf_mbox graf_movebox                  /* Wrong GEM binding           */
#define graf_rubbox graf_rubberbox              /* Wrong GEM binding           */
#endif /* LATTICE_C */

#if TURBO_C
#define graf_mbox graf_movebox                  /* Wrong GEM binding           */
#define graf_rubbox graf_rubberbox              /* Wrong GEM binding           */
#endif /* TURBO_C */

#if MW_C
#define VOID WORD                               /* MW_C doesn't know (void *)  */
#endif /* MW_C */

#if LATTICE_C
#define ADR(A) (LONG)A >> 16, (LONG)A & 0xFFFF
#else
#define ADR(A) (WORD)((LONG)A >> 16), (WORD)((LONG)A & 0xFFFF)
#endif /* LATTICE_C */
#endif /* GEMDOS */
```

```
#if MSDOS | OS2                                    /* MSDOS or OS2 compilers      *
#define ADR(A) (WORD)((LONG)A & 0xFFFF), (WORD)((LONG)A >> 16)
#endif /* MSDOS */

#if FLEXOS                                         /* FlexOS compilers            *
#define ADR(A) (WORD)((LONG)A & 0xFFFF), (WORD)((LONG)A >> 16)
#endif /* FLEXOS */

#if MS_C | TURBO_C | HIGH_C                        /* ANSI compilers              *
#define ANSI 1
#define _(params) params                           /* parameter checking          *
#else
#define ANSI 0
#define _(params) ()                               /* no parameter checking       *
#define const
#define volatile
#define size_t UINT
#endif

#if DR_C | LASER_C | LATTICE_C | MW_C | HIGH_C
#define cdecl
#define pascal
#endif

#define CONST    const
#define VOLATILE volatile
#define CDECL    cdecl
#define PASCAL   pascal

#define SIZE_T   size_t

#ifndef VOID
#define VOID     void
#endif

/*****************************************************************************/
/* OPERATING SYSTEM DEPENDENT DEFINITIONS                                    */
/*****************************************************************************/

#if GEMDOS
#define FAR                                        /* Far Pointer                 *
#define NEAR                                       /* Near Pointer                *
#else
#define FAR      far                               /* Far Pointer                 *
#define NEAR     near                              /* Near Pointer                *
#endif /* GEMDOS */
```

```
#if GEM & GEM1
#define appl_bvset(bvdisk, bvhard)
#define appl_yield() evnt_timer (0, 0)
#define menu_unregister(mid)
#define scrp_clear()
#define xgrf_stepcalc(orgw, orgh, xc, yc, w, h, pcx, pcy, pcnt, pxstep, pystep)
#define xgrf_2box(xc, yc, w, h, corners, cnt, xstep, ystep, doubled)
#endif /* GEM1 */

#if GEM & (GEM1 | XGEM)
#define shel_rdef(lpcmd, lpdir)
#define shel_wdef(lpcmd, lpdir)
#endif /* GEM1 | XGEM */

#if GEM & (GEM1 | GEM2)
#define menu_click(click, setit)
#endif /* GEM1 | GEM2 */

#if GEM & (GEM2 | GEM3 | XGEM)
#define fsel_exinput(pipath, pisel, pbutton, plabel)\
        fsel_input  (pipath, pisel, pbutton)
#define wind_new()
#endif /* GEM2 | GEM3 | XGEM */

/*****************************************************************************/
/* MISCELLANEOUS DEFINITIONS                                                 */
/*****************************************************************************/

#define FALSE   (BOOLEAN)0                  /* Function FALSE value          */
#define TRUE    (BOOLEAN)1                  /* Function TRUE  value          */
#define FAILURE (-1)                        /* Function failure return val */
#define SUCCESS 0                           /* Function success return val */
#define FOREVER for (;;)                    /* Infinite loop declaration     */
#define EOS     '\0'                        /* End of string value           */

#ifndef NULL
#define NULL    0L                          /* Null long value               */
#endif

#ifndef EOF
#define EOF     (-1)                        /* EOF value                     */
#endif

#endif /* __PORTAB__ */
```

## Header-Datei AES.H:

```
/*************************************************************************
/*   AES.H: Common AES definitions and structures.                       *
/*************************************************************************
/*                                                                       *
/*   Authors: Dieter & Juergen Geiss                                     *
/*                                                                       *
/*************************************************************************

#ifndef __AES__
#define __AES__

/****** GEMparams ***************************************************************

#if GEMDOS
#if TURBO_C

typedef struct
{
  WORD contrl [11];
  WORD global [80];
  WORD intin [128];
  WORD ptsin [128];
  WORD intout [45];
  WORD ptsout [128];
  VOID FAR *addrin [128];
  VOID FAR *addrout [128];
} GEMPARBLK;

extern GEMPARBLK _GemParBlk;

#endif /* TURBO_C */
#endif /* GEMDOS */

/****** Application library *****************************************************

WORD  appl_init       _((VOID));
WORD  appl_read       _((WORD rwid, WORD length, VOID FAR *pbuff));
WORD  appl_write      _((WORD rwid, WORD length, VOID FAR *pbuff));
WORD  appl_find       _((BYTE FAR *pname));
WORD  appl_tplay      _((VOID FAR *tbuffer, WORD tlenght, WORD tscale));
WORD  appl_trecord    _((VOID FAR *tbuffer, WORD tlength));
```

```
#if GEM & (GEM2 | GEM3 | XGEM)
WORD  appl_bvset      _((UWORD bvdisk, UWORD bvhard));
VOID  appl_yield      _((VOID));
#endif /* GEM2 | GEM3 | XGEM */

WORD  appl_exit       _((VOID));

/****** Event library *************************************************************/

typedef struct orect
{
  struct orect FAR *o_link;
  WORD          o_x;
  WORD          o_y;
  WORD          o_w;
  WORD          o_h;
} ORECT;

typedef struct grect
{
  WORD g_x;
  WORD g_y;
  WORD g_w;
  WORD g_h;
} GRECT;

typedef struct mevent
{
  UWORD e_flags;
  UWORD e_bclk;
  UWORD e_bmsk;
  UWORD e_bst;
  UWORD e_m1flags;
  GRECT e_m1;
  UWORD e_m2flags;
  GRECT e_m2;
  WORD  *e_mepbuf;
  ULONG e_time;
  WORD  e_mx;
  WORD  e_my;
  UWORD e_mb;
  UWORD e_ks;
  UWORD e_kr;
  UWORD e_br;
  UWORD e_m3flags;
  GRECT e_m3;
```

```
  WORD  e_xtra0;
  WORD  *e_smepbuf;
  ULONG e_xtra1;
  ULONG e_xtra2;
] MEVENT;

/* multi flags */

#define MU_KEYBD    0x0001
#define MU_BUTTON   0x0002
#define MU_M1       0x0004
#define MU_M2       0x0008
#define MU_MESAG    0x0010
#define MU_TIMER    0x0020
#define MU_M3       0x0040
#define MU_SYSMESAG 0x0080
#define MU_POSTEV   0x1000

/* keyboard states */

#define K_RSHIFT    0x0001
#define K_LSHIFT    0x0002
#define K_CTRL      0x0004
#define K_ALT       0x0008

/* message values */

#define SCR_MGR     0x0001 /* process id of the screen manager */

#define AP_MSG           0

#define MN_SELECTED     10
#define WM_REDRAW       20
#define WM_TOPPED       21
#define WM_CLOSED       22
#define WM_FULLED       23
#define WM_ARROWED      24
#define WM_HSLID        25
#define WM_VSLID        26
#define WM_SIZED        27
#define WM_MOVED        28
#define WM_NEWTOP       29 /* for compatibility */
#define WM_UNTOPPED     30
#define WM_ONTOP        31
#define WM_OFFTOP       32
#define PR_FINISH       33
```

```
#define AC_OPEN         40
#define AC_CLOSE        41

#define CT_UPDATE       50
#define CT_MOVE         51
#define CT_NEWTOP       52
#define CT_SWITCH       53

#define SV_ONTOP       110
#define SV_OFFTOP      111

typedef struct
{
  WORD m_out;
  WORD m_x;
  WORD m_y;
  WORD m_w;
  WORD m_h;
} MOBLK;

UWORD evnt_keybd      _((VOID));
WORD  evnt_button     _((WORD clicks, UWORD mask, UWORD state,
                        WORD FAR *pmx, WORD FAR *pmy,
                        WORD FAR *pmb, WORD FAR *pks));
WRD   evnt_mouse      _((WORD flags, WORD x, WORD y, WORD width, WORD height,
                        WORD FAR *pmx, WORD FAR *pmy,
                        WORD FAR *pmb, WORD FAR *pks));
WORD  evnt_mesag      _((WORD FAR *pbuff));
WORD  evnt_timer      _((UWORD locnt, UWORD hicnt));
WORD  evnt_multi      _((UWORD flags, UWORD bclk, UWORD bmsk, UWORD bst,
                        UWORD m1flags,
                        UWORD m1x, UWORD m1y, UWORD m1w, UWORD m1h,
                        UWORD m2flags,
                        UWORD m2x, UWORD m2y, UWORD m2w, UWORD m2h,
                        WORD FAR *mepbuff, UWORD tlc, UWORD thc,
                        WORD FAR *pmx, WORD FAR *pmy, WORD FAR *pmb,
                        WORD FAR *pks, UWORD FAR *pkr, WORD FAR *pbr));
#if GEM & XGEM
WORD  evnt_event      _((MEVENT *pmevent));
#endif /* XGEM */
WORD  evnt_dclick     _((WORD rate, WORD setit));
```

```
/****** Object structure ***************************************************/

typedef struct object
{
  WORD  ob_next;   /* -> object's next sibling       */
  WORD  ob_head;   /* -> head of object's children   */
  WORD  ob_tail;   /* -> tail of object's children   */
  UWORD ob_type;   /* type of object- BOX, CHAR, ... */
  UWORD ob_flags;  /* flags                          */
  UWORD ob_state;  /* state- SELECTED, CROSSED, ...  */
  LONG  ob_spec;   /* "out"- -> anything else        */
  WORD  ob_x;      /* upper left corner of object    */
  WORD  ob_y;      /* upper left corner of object    */
  WORD  ob_width;  /* width of object                */
  WORD  ob_height; /* height of object               */
} OBJECT;

/****** Menu library *******************************************************/

WORD  menu_bar        _((OBJECT FAR *tree, WORD showit));
WORD  menu_icheck     _((OBJECT FAR *tree, WORD itemnum, WORD checkit));
WORD  menu_ienable    _((OBJECT FAR *tree, WORD itemnum, WORD enableit));
WORD  menu_tnormal    _((OBJECT FAR *tree, WORD titlenum, WORD normalit));
WORD  menu_text       _((OBJECT FAR *tree, WORD inum, BYTE FAR *ptext));
WORD  menu_register   _((WORD pid, BYTE FAR *pstr));

#if GEM & (GEM2 | GEM3 | XGEM)
WORD  menu_unregister _((WORD mid));
#endif /* GEM2 | GEM3 | XGEM */

#if GEM & (GEM3 | XGEM)
WORD  menu_click      _((WORD click, WORD setit));
#endif /* GEM3 | XGEM */

/****** Object library *****************************************************/

#define OB_NEXT(tree, id)    tree [id].ob_next
#define OB_HEAD(tree, id)    tree [id].ob_head
#define OB_TAIL(tree, id)    tree [id].ob_tail
#define OB_TYPE(tree, id)    (tree [id].ob_type & 0xff)
#define OB_EXTYPE(tree, id)  (tree [id].ob_type >> 8)
#define OB_FLAGS(tree, id)   tree [id].ob_flags
#define OB_STATE(tree, id)   tree [id].ob_state
#define OB_SPEC(tree, id)    tree [id].ob_spec
#define OB_X(tree, id)       tree [id].ob_x
#define OB_Y(tree, id)       tree [id].ob_y
#define OB_WIDTH(tree, id)   tree [id].ob_width
#define OB_HEIGHT(tree, id)  tree [id].ob_height
```

```
#define ROOT            0
#define NIL            -1 /* nil object index */
#define MAX_LEN        81 /* max string length */
#define MAX_DEPTH       8 /* max depth of search or draw for objects */

#define IP_HOLLOW       0 /* inside patterns */
#define IP_1PATT        1
#define IP_2PATT        2
#define IP_3PATT        3
#define IP_4PATT        4
#define IP_5PATT        5
#define IP_6PATT        6
#define IP_SOLID        7
                          /* system foreground and */
                          /*   background rules    */
#define SYS_FG     0x1100 /*   but transparent     */

#define WTS_FG     0x11a1 /* window title selected */
                          /*   using pattern 2 &   */
                          /*   replace mode text   */
#define WTN_FG     0x1100 /* window title normal   */

#define IBM             3 /* font types */
#define SMALL           5

#define G_BOX          20 /* graphic types of obs */
#define G_TEXT         21
#define G_BOXTEXT      22
#define G_IMAGE        23
#define G_USERDEF      24
#define G_PROGDEF      24 /* for compatibility */
#define G_IBOX         25
#define G_BUTTON       26
#define G_BOXCHAR      27
#define G_STRING       28
#define G_FTEXT        29
#define G_FBOXTEXT     30
#define G_ICON         31
#define G_TITLE        32

#define NONE       0x0000 /* Object flags */
#define SELECTABLE 0x0001
#define DEFAULT    0x0002
#define EXIT       0x0004
#define EDITABLE   0x0008
#define RBUTTON    0x0010
```

```
#define LASTOB      0x0020
#define TOUCHEXIT   0x0040
#define HIDETREE    0x0080
#define INDIRECT    0x0100


#define NORMAL      0x0000 /* Object states */
#define SELECTED    0x0001
#define CROSSED     0x0002
#define CHECKED     0x0004
#define DISABLED    0x0008
#define OUTLINED    0x0010
#define SHADOWED    0x0020
#define WHITEBAK    0x0040
#define DRAW3D      0x0080


#ifndef WHITE
#define WHITE           0 /* Object colors */
#define BLACK           1
#define RED             2
#define GREEN           3
#define BLUE            4
#define CYAN            5
#define YELLOW          6
#define MAGENTA         7
#define DWHITE          8
#define DBLACK          9
#define DRED           10
#define DGREEN         11
#define DBLUE          12
#define DCYAN          13
#define DYELLOW        14
#define DMAGENTA       15
#endif


#define EDSTART         0 /* editable text field definitions */
#define EDINIT          1
#define EDCHAR          2
#define EDEND           3


#define TE_LEFT         0 /* editable text justification */
#define TE_RIGHT        1
#define TE_CNTR         2
```

```
typedef struct text_edinfo
{
  BYTE FAR *te_ptext;       /* ptr to text (must be 1st)     */
  BYTE FAR *te_ptmplt;      /* ptr to template               */
  BYTE FAR *te_pvalid;      /* ptr to validation chrs.       */
  WORD te_font;             /* font                          */
  WORD te_junk1;            /* junk word                     */
  WORD te_just;             /* justification- left, right... */
  WORD te_color;            /* color information word        */
  WORD te_junk2;            /* junk word                     */
  WORD te_thickness;        /* border thickness              */
  WORD te_txtlen;           /* length of text string         */
  WORD te_tmplen;           /* length of template string     */
} TEDINFO;



typedef struct icon_block
{
  WORD FAR *ib_pmask;   /* ptr to mask of icon                */
  WORD FAR *ib_pdata;   /* ptr to data of icon                */
  BYTE FAR *ib_ptext;   /* ptr to text of icon                */
  WORD ib_char;         /* character in icon                  */
  WORD ib_xchar;        /* x-coordinate of ib_char            */
  WORD ib_ychar;        /* y-coordinate of ib_char            */
  WORD ib_xicon;        /* x-coordinate of icon               */
  WORD ib_yicon;        /* y-coordinate of icon               */
  WORD ib_wicon;        /* width of icon in pixels            */
  WORD ib_hicon;        /* height of icon in pixels           */
  WORD ib_xtext;        /* x-coordinate of the icon's text    */
  WORD ib_ytext;        /* y-coordinate of the icon's text    */
  WORD ib_wtext;        /* width of rectangle for icon's text */
  WORD ib_htext;        /* height of icon's text in pixels    */
} ICONBLK;



typedef struct bit_block
{
  WORD FAR *bi_pdata;   /* ptr to bit forms data  */
  WORD bi_wb;           /* width of form in bytes */
  WORD bi_hl;           /* height in scan lines   */
  WORD bi_x;            /* source x in bit form   */
  WORD bi_y;            /* source y in bit form   */
  WORD bi_color;        /* fg color of blt        */
} BITBLK;
```

```
typedef struct parm_blk
{
  OBJECT FAR *pb_tree;                /* ptr to obj tree for user defined obj */
  WORD   pb_obj;                      /* index of user defined object         */
  WORD   pb_prevstate;                /* old state to be changed              */
  WORD   pb_currstate;                /* changed (new) state of object        */
  WORD   pb_x, pb_y, pb_w, pb_h;      /* location of object on screen         */
  WORD   pb_xc, pb_yc, pb_wc, pb_hc; /* current clipping rectangle on screen */
  LONG   pb_parm;                     /* same as ub_parm in USERBLK struct    */
} PARMBLK;


typedef struct user_blk
{
#if MSDOS
  WORD (FAR *ub_code) _((VOID));              /* pointer to drawing function */
#else
  WORD CDECL (FAR *ub_code) _((PARMBLK *pb)); /* pointer to drawing function */
#endif
  LONG ub_parm;                               /* parm for drawing function   */
} USERBLK;


typedef struct appl_blk /* for compatibility */
{
#if MSDOS
  WORD (FAR *ab_code) _((VOID));              /* pointer to drawing function */
#else
  WORD CDECL (FAR *ab_code) _((PARMBLK *pb)); /* pointer to drawing function */
#endif
  LONG ab_parm;                               /* parm for drawing function   */
} APPLBLK;


WORD  objc_add        _((OBJECT FAR *tree, WORD parent, WORD child));
WORD  objc_delete     _((OBJECT FAR *tree, WORD delob));
WORD  objc_draw       _((OBJECT FAR *tree, WORD drawob, WORD depth,
                         WORD xc, WORD yc, WORD wc, WORD hc));
WORD  objc_find       _((OBJECT FAR *tree, WORD startob, WORD depth,
                         WORD mx, WORD my));
WORD  objc_offset     _((OBJECT FAR *tree, WORD obj, WORD FAR *poffx,
                         WORD FAR *poffy));
WORD  objc_order      _((OBJECT FAR *tree, WORD mov_obj, WORD newpos));
WORD  objc_edit       _((OBJECT FAR *tree, WORD obj, WORD inchar,
                         WORD FAR *idx, WORD kind));
WORD  objc_change     _((OBJECT FAR *tree, WORD drawob, WORD depth,
                         WORD xc, WORD yc, WORD wc, WORD hc,
                         WORD nestate, WORD redraw));
```

```
/****** Form library *********************************************************/

/* Form flags */

#define FMD_START         0
#define FMD_GROW          1
#define FMD_SHRINK        2
#define FMD_FINISH        3
#define FMD_ASTART        4
#define FMD_AFINISH       5

WORD  form_do          _((OBJECT FAR *form, WORD start));
WORD  form_dial        _((WORD dtype, WORD ix, WORD iy, WORD iw, WORD ih,
                          WORD x, WORD y, WORD w, WORD h));
WORD  form_alert       _((WORD defbut, BYTE FAR *astring));
WORD  form_error       _((WORD errnum));
WORD  form_center      _((OBJECT FAR *tree, WORD FAR *pcx, WORD FAR *pcy,
                          WORD FAR *pcw, WORD FAR *pch));
WORD  form_keybd       _((OBJECT FAR *form, WORD obj, WORD nxt_obj,
                          UWORD thechar, WORD FAR *pnxt_obj, UWORD FAR *pchar));
WORD  form_button      _((OBJECT FAR *form, WORD obj, WORD clks,
                          WORD FAR *pnxt_obj));

/****** Graphics library *****************************************************

/* Mouse Forms */

#define ARROW             0
#define TEXT_CRSR         1
#define HOURGLASS         2
#define BUSY_BEE          2 /* for compatibility */
#define POINT_HAND        3
#define FLAT_HAND         4
#define THIN_CROSS        5
#define THICK_CROSS       6
#define OUTLN_CROSS       7
#define M_PUSH          100
#define M_POP           101
#define USER_DEF        255
#define M_OFF           256
#define M_ON            257
```

```
/* Mouse Form Definition Block */

typedef struct mfstr
{
  WORD mf_xhot;
  WORD mf_yhot;
  WORD mf_nplanes;
  WORD mf_fg;
  WORD mf_bg;
  WORD mf_mask [16];
  WORD mf_data [16];
} MFORM;

WORD  graf_rubbox     _((WORD xorigin, WORD yorigin, WORD wmin, WORD hmin,
                        WORD FAR *pwend, WORD FAR *phend));
WORD  graf_dragbox    _((WORD w, WORD h, WORD sx, WORD sy, WORD xc, WORD yc,
                        WORD wc, WORD hc, WORD FAR *pdx, WORD FAR *pdy));
WORD  graf_mbox       _((WORD w, WORD h, WORD srcx, WORD srcy,
                        WORD dstx, WORD dsty));
VOID  graf_growbox    _((WORD stx, WORD sty, WORD stw, WORD sth,
                        WORD finx, WORD finy, WORD finw, WORD finh));
VOID  graf_shrinkbox  _((WORD finx, WORD finy, WORD finw, WORD finh,
                        WORD stx, WORD sty, WORD stw, WORD sth));
WORD  graf_watchbox   _((OBJECT FAR *tree, WORD obj, WORD instate,
                        WORD outstate));
WORD  graf_slidebox   _((OBJECT FAR *tree, WORD parent, WORD obj,
                        WORD isvert));
WORD  graf_handle     _((WORD FAR *pwchar, WORD FAR *phchar,
                        WORD FAR *pwbox, WORD FAR *phbox));
WORD  graf_mouse      _((WORD m_number, MFORM FAR *m_addr));
VOID  graf_mkstate    _((WORD FAR *pmx, WORD FAR *gpmy,
                        WORD FAR *pmstate, WORD FAR *pkstate));

/****** Scrap library ***************************************************

#define SCRAP_CSV   0x0001
#define SCRAP_TXT   0x0002
#define SCRAP_GEM   0x0004
#define SCRAP_IMG   0x0008
#define SCRAP_DCA   0x0010
#define SCRAP_USR   0x8000

WORD  scrp_read       _((BYTE FAR *pscrap));
WORD  scrp_write      _((BYTE FAR *pscrap));

#if GEM & (GEM2 | GEM3 | XGEM)
WORD  scrp_clear      _((VOID));
#endif /* GEM2 | GEM3 | XGEM*/
```

```
/****** File selector library ***********************************************/

WORD  fsel_input      _((BYTE FAR *pipath, BYTE FAR *pisel,
                        WORD FAR *pbutton));

#if GEM & GEM1
WORD  fsel_exinput    _((BYTE FAR *pipath, BYTE FAR *pisel, WORD FAR *pbutton,
                        BYTE *plabel));
                        #endif /* GEM1 */

/****** Window library ******************************************************/

/* Window Attributes */

#define NAME         0x0001
#define CLOSER       0x0002
#define FULLER       0x0004
#define MOVER        0x0008
#define INFO         0x0010
#define SIZER        0x0020
#define UPARROW      0x0040
#define DNARROW      0x0080
#define VSLIDE       0x0100
#define LFARROW      0x0200
#define RTARROW      0x0400
#define HSLIDE       0x0800
#define HOTCLOSEBOX 0x1000 /* for compatibility */
#define HOTCLOSE     0x1000

/* wind_calc flags */

#define WC_BORDER        0
#define WC_WORK          1

/* wind_get/wind_set flags */

#define WF_KIND          1
#define WF_NAME          2
#define WF_INFO          3
#define WF_WXYWH         4
#define WF_WORKXYWH      4 /* for compatibility */
#define WF_CXYWH         5
#define WF_CURRXYWH      5 /* for compatibility */
#define WF_PXYWH         6
#define WF_PREVXYWH      6 /* for compatibility */
#define WF_FXYWH         7
```

```
#define WF_FULLXYWH      7 /* for compatibility */
#define WF_HSLIDE        8
#define WF_VSLIDE        9
#define WF_TOP          10
#define WF_FIRSTXYWH    11
#define WF_NEXTXYWH     12
#define WF_IGNORE       13
#define WF_NEWDESK      14
#define WF_HSLSIZE      15
#define WF_VSLSIZE      16
#define WF_SCREEN       17
#define WF_TATTRB       18
#define WF_SIZTOP       19
#define WF_TOPAP        20

/* update flags */

#define END_UPDATE       0
#define BEG_UPDATE       1
#define END_MCTRL        2
#define BEG_MCTRL        3
#define BEG_EMERG        4
#define END_EMERG        5

/* arrow message */

#define WA_SUBWIN        1
#define WA_KEEPWIN       2

#define WA_UPPAGE        0 /* Window Arrow Up Page    */
#define WA_DNPAGE        1 /* Window Arrow Down Page  */
#define WA_UPLINE        2 /* Window Arrow Up Line    */
#define WA_DNLINE        3 /* Window Arrow Down Line  */
#define WA_LFPAGE        4 /* Window Arrow Left Page  */
#define WA_RTPAGE        5 /* Window Arrow Right Page */
#define WA_LFLINE        6 /* Window Arrow Left Line  */
#define WA_RTLINE        7 /* Window Arrow Right Line */

WORD  wind_create     _((UWORD kind, WORD wx, WORD wy, WORD ww, WORD wh));
WORD  wind_open       _((WORD handle, WORD wx, WORD wy, WORD ww, WORD wh));
WORD  wind_close      _((WORD handle));
WORD  wind_delete     _((WORD handle));
#if GEMDOS
WORD  wind_get        _((WORD w_handle, WORD wfield, ...));
WORD  wind_set        _((WORD w_handle, WORD wfield, ...));
#else
```

```
WORD  wind_get        _((WORD w_handle, WORD wfield,
                        WORD *pw1, WORD *pw2, WORD *pw3, WORD *pw4));
WORD  wind_set        _((WORD w_handle, WORD wfield,
                        WORD w1, WORD w2, WORD w3, WORD w4));
#endif
WORD  wind_find       _((WORD mx, WORD my));
WORD  wind_update     _((WORD beg_update));
WORD  wind_calc       _((WORD wctype, UWORD kind,
                        WORD x, WORD y, WORD w, WORD h,
                        WORD FAR *px, WORD FAR *py,
                        WORD FAR *pw, WORD FAR *ph));

#if GEM & GEM1
VOID  wind_new        _((VOID));
#endif /* GEM1 */


#if GEM & XGEM
WORD  wind_apfind     _((WORD mx, WORD my));
#endif /* XGEM */

/****** Resource library ************************************************/

/* data structure types */


#define R_TREE          0
#define R_OBJECT        1
#define R_TEDINFO       2
#define R_ICONBLK       3
#define R_BITBLK        4
#define R_STRING        5 /* gets pointer to free strings     */
#define R_IMAGEDATA     6 /* gets pointer to free images      */
#define R_OBSPEC        7
#define R_TEPTEXT       8 /* sub ptrs in TEDINFO              */
#define R_TEPTMPLT      9
#define R_TEPVALID     10
#define R_IBPMASK      11 /* sub ptrs in ICONBLK              */
#define R_IBPDATA      12
#define R_IBPTEXT      13
#define R_BIPDATA      14 /* sub ptrs in BITBLK               */
#define R_FRSTR        15 /* gets addr of ptr to free strings */
#define R_FRIMG        16 /* gets addr of ptr to free images  */


/* used in RSCREATE.C */
```

```
typedef struct rshdr
{
  UWORD rsh_vrsn;
  UWORD rsh_object;
  UWORD rsh_tedinfo;
  UWORD rsh_iconblk;        /* list of ICONBLKS          */
  UWORD rsh_bitblk;
  UWORD rsh_frstr;
  UWORD rsh_string;
  UWORD rsh_imdata;         /* image data                */
  UWORD rsh_frimg;
  UWORD rsh_trindex;
  UWORD rsh_nobs;           /* counts of various structs */
  UWORD rsh_ntree;
  UWORD rsh_nted;
  UWORD rsh_nib;
  UWORD rsh_nbb;
  UWORD rsh_nstring;
  UWORD rsh_nimages;
  UWORD rsh_rssize;         /* total bytes in resource   */
} RSHDR;

#define F_ATTR 0            /* file attr for dos_create  */

WORD  rsrc_load       _((BYTE FAR *rsname));
WORD  rsrc_free       _((VOID));
WORD  rsrc_gaddr      _((WORD rstype, WORD rsid, VOID FAR *paddr));
WORD  rsrc_saddr      _((WORD rstype, WORD rsid, VOID FAR *lngval));
WORD  rsrc_obfix      _((OBJECT FAR *tree, WORD obj));

/****** Shell library ***********************************************************

WORD  shel_read       _((BYTE FAR *pcmd, BYTE FAR *ptail));
WORD  shel_write      _((WORD doex, WORD isgr, WORD isover, BYTE FAR *pcmd,
                         BYTE FAR *ptail));
WORD  shel_get        _((BYTE FAR *addr, WORD len));
WORD  shel_put        _((BYTE FAR *addr, WORD len));
WORD  shel_find       _((BYTE FAR *ppath));
WORD  shel_envrn      _((BYTE FAR * FAR *ppath, BYTE FAR *psrch));

#if GEM & (GEM2 | GEM3)
WORD  shel_rdef       _((BYTE FAR *lpcmd, BYTE FAR *lpdir));
WORD  shel_wdef       _((BYTE FAR *lpcmd, BYTE FAR *lpdir));
#endif /* GEM2 | GEM3 */
```

```
/****** Extended graphics library ********************************************/

#if GEM & (GEM2 | GEM3 | XGEM)
WORD  xgrf_stepcalc   _((WORD orgw, WORD orgh, WORD xc, WORD yc,
                         WORD w, WORD h, WORD FAR *pcx, WORD FAR *pcy,
                         WORD FAR *pcnt, WORD FAR *pxstep, WORD FAR *pystep));
WORD  xgrf_2box       _((WORD xc, WORD yc, WORD w, WORD h, WORD corners,
                         WORD cnt, WORD xstep, WORD ystep, WORD doubled));
#endif /* GEM2 | GEM3 | XGEM */

/*****************************************************************************/

#endif /* __AES__ */
```

## Header-Datei VDI.H:

```
/*****************************************************************************/
/*   VDI.H: Common VDI definitions and structures.                           */
/*****************************************************************************/
/*                                                                           */
/*   Authors: Dieter & Juergen Geiss                                         */
/*                                                                           */
/*****************************************************************************/

#ifndef __VDI__
#define __VDI__

/****** Control library ******************************************************/

#define OW_FILE          0 /* output device type in the low-order byte */
#define OW_SERIAL        1
#define OW_PARALLEL      2
#define OW_DEVICE        3
#define OW_NOCHANGE    255

#define OW_LETTER1       0 /* page size index in the high-order byte */
#define OW_HALF          5
#define OW_B5           10
#define OW_LETTER2      20
#define OW_A4           30
#define OW_LEGAL        40
#define OW_DOUBLE       50
#define OW_BROAD        55
#define OW_INDIRECT    255 /* use work_in [101] and work_in [102] */
```

```
VOID v_get_driver_info _((WORD device_id, WORD info_select,
                          UBYTE FAR *info_string));
VOID v_opnwk           _((WORD FAR *work_in, WORD FAR *handle,
                          WORD FAR *work_out));
VOID v_clswk           _((WORD handle));
VOID v_clrwk           _((WORD handle));
VOID v_updwk           _((WORD handle));
VOID v_opnvwk          _((WORD FAR *work_in, WORD FAR *handle,
                          WORD FAR *work_out));
VOID v_clsvwk          _((WORD handle));
WORD vst_load_fonts    _((WORD handle, WORD select));
VOID vst_unload_fonts  _((WORD handle, WORD select));
WORD vst_ex_load_fonts _((WORD handle, WORD select, WORD font_max,
                          WORD font_free));
VOID vs_clip           _((WORD handle, WORD clip_flag, WORD FAR *pxyarray));
VOID v_set_app_buff    _((VOID FAR *address, WORD nparagraphs));
WORD v_bez_on          _((WORD handle));
WORD v_bez_off         _((WORD handle));
WORD v_bez_qual        _((WORD handle, WORD prcnt));
VOID v_pat_rotate      _((WORD handle, WORD angle));

/****** Output library ************************************************

VOID v_pline           _((WORD handle, WORD count, WORD FAR *xy));
VOID v_pmarker         _((WORD handle, WORD count, WORD FAR *xy));
VOID v_gtext           _((WORD handle, WORD x, WORD y, BYTE FAR *string));
VOID v_etext           _((WORD handle, WORD x, WORD y, UBYTE FAR *string,
                          WORD FAR *offsets));
VOID v_fillarea        _((WORD handle, WORD count, WORD FAR *xy));
VOID v_cellarray       _((WORD handle, WORD FAR *pxyarray, WORD row_length,
                          WORD el_used, WORD num_rows, WORD wrt_mode,
                          WORD FAR *colarray));
VOID v_bar             _((WORD handle, WORD FAR *pxyarray));
VOID v_arc             _((WORD handle, WORD x, WORD y, WORD radius,
                          WORD begang, WORD endang));
VOID v_pieslice        _((WORD handle, WORD x, WORD y, WORD radius,
                          WORD begang, WORD endang));
VOID v_circle          _((WORD handle, WORD x, WORD y, WORD radius));
VOID v_ellipse         _((WORD handle, WORD x, WORD y, WORD xradius,
                          WORD yradius));
VOID v_ellarc          _((WORD handle, WORD x, WORD y, WORD xradius,
                          WORD yradius, WORD begang, WORD endang));
VOID v_ellpie          _((WORD handle, WORD x, WORD y, WORD xradius,
                          WORD yradius, WORD begang, WORD endang));
VOID v_rbox            _((WORD handle, WORD FAR *xyarray));
VOID v_rfbox           _((WORD handle, WORD FAR *xyarray));
```

```
VOID v_justified        _((WORD handle, WORD x, WORD y, BYTE FAR *string,
                           WORD length, WORD word_space, WORD char_space));
VOID v_contourfill      _((WORD handle, WORD x, WORD y, WORD index));
VOID vr_recfl           _((WORD handle, WORD FAR *pxyarray));
VOID v_bez              _((WORD handle, WORD count, WORD FAR *xyarr,
                           UBYTE FAR *bezarr, WORD FAR *minmax, WORD FAR *npts,
                           WORD FAR *nmove));
VOID v_bezfill          _((WORD handle, WORD count, WORD FAR *xyarr,
                           UBYTE FAR *bezarr, WORD FAR *minmax, WORD FAR *npts,
                           WORD FAR *nmove));

/****** Attribute library ************************************************/

#define MD_REPLACE       1 /* gsx modes */
#define MD_TRANS         2
#define MD_XOR           3
#define MD_ERASE         4

#define FIS_HOLLOW       0 /* gsx styles */
#define FIS_SOLID        1
#define FIS_PATTERN      2
#define FIS_HATCH        3
#define FIS_USER         4

#define ALL_WHITE        0 /* bit blt rules */
#define S_AND_D          1
#define S_AND_NOTD       2
#define S_ONLY           3
#define NOTS_AND_D       4
#define D_ONLY           5
#define S_XOR_D          6
#define S_OR_D           7
#define NOT_SORD         8
#define NOT_SXORD        9
#define D_INVERT        10
#define S_OR_NOTD       11
#define NOT_D           12
#define NOTS_OR_D       13
#define NOT_SANDD       14
#define ALL_BLACK       15

#ifndef WHITE
#define WHITE            0 /* colors */
#define BLACK            1
#define RED              2
#define GREEN            3
```

```
#define BLUE            4
#define CYAN            5
#define YELLOW          6
#define MAGENTA         7
#define DWHITE          8
#define DBLACK          9
#define DRED           10
#define DGREEN         11
#define DBLUE          12
#define DCYAN          13
#define DYELLOW        14
#define DMAGENTA       15
#endif

#define SOLID           1 /* line types */
#define LONGDASH        2
#define DOT             3
#define DASHDOT         4
#define DASH            5
#define DASH2DOT        6
#define USERLINE        7

#define SQUARED         0 /* line ends */
#define ARROWED         1
#define ROUNDED         2

#define PM_DOT          1 /* polymarker types */
#define PM_PLUS         2
#define PM_ASTERISK     3
#define PM_SQUARE       4
#define PM_DIAGCROSS    5
#define PM_DIAMOND      6

#define TXT_NORMAL      0x0000 /* text effects */
#define TXT_THICKENED   0x0001
#define TXT_LIGHT       0x0002
#define TXT_SKEWED      0x0004
#define TXT_UNDERLINED  0x0008
#define TXT_OUTLINED    0x0010
#define TXT_SHADOWED    0x0020

#define ALI_LEFT        0 /* horizontal text alignment */
#define ALI_CENTER      1
#define ALI_RIGHT       2
```

```
#define ALI_BASE        0 /* vertical text alignment */
#define ALI_HALF        1
#define ALI_ASCENT      2
#define ALI_BOTTOM      3
#define ALI_DESCENT     4
#define ALI_TOP         5


/* fill pattern */


typedef struct patarray
{
  WORD patword [16];
} FILLPAT;


WORD vswr_mode          _((WORD handle, WORD mode));
VOID vs_color           _((WORD handle, WORD index, WORD FAR *rgb_in));
WORD vsl_type           _((WORD handle, WORD style));
VOID vsl_udsty          _((WORD handle, WORD pattern));
WORD vsl_width          _((WORD handle, WORD width));
WORD vsl_color          _((WORD handle, WORD color_index));
VOID vsl_ends           _((WORD handle, WORD beg_style, WORD end_style));
WORD vsm_type           _((WORD handle, WORD symbol));
WORD vsm_height         _((WORD handle, WORD height));
WORD vsm_color          _((WORD handle, WORD color_index));
VOID vst_height         _((WORD handle, WORD height,
                           WORD FAR *char_width, WORD FAR *char_height,
                           WORD FAR *cell_width, WORD FAR *cell_height));
WORD vst_point          _((WORD handle, WORD point,
                           WORD FAR *char_width, WORD FAR *char_height,
                           WORD FAR *cell_width, WORD FAR *cell_height));
WORD vst_rotation       _((WORD handle, WORD angle));
WORD vst_font           _((WORD handle, WORD font));
WORD vst_color          _((WORD handle, WORD color_index));
WORD vst_effects        _((WORD handle, WORD effect));
VOID vst_alignment      _((WORD handle, WORD hor_in, WORD vert_in,
                           WORD FAR *hor_out, WORD FAR *vert_out));
WORD vsf_interior       _((WORD handle, WORD style));
WORD vsf_style          _((WORD handle, WORD style_index));
WORD vsf_color          _((WORD handle, WORD color_index));
WORD vsf_perimeter      _((WORD handle, WORD per_vis, WORD per_style));
VOID vsf_udpat          _((WORD handle, WORD FAR *pfill_pat, WORD planes));
VOID vs_grayoverride    _((WORD handle, WORD grayval));
```

```
/****** Raster library ***********************************************************

/* Memory Form Definition Block */

typedef struct memform
{
  VOID FAR *mp;
  WORD fwp;
  WORD fh;
  WORD fww;
  WORD ff;
  WORD np;
  WORD r1;
  WORD r2;
  WORD r3;
} MFDB;

typedef struct fdbstr   /* for compatibility */
{
  VOID FAR *fd_addr;
  WORD fd_w;
  WORD fd_h;
  WORD fd_wdwidth;
  WORD fd_stand;
  WORD fd_nplanes;
  WORD fd_r1;
  WORD fd_r2;
  WORD fd_r3;
} FDB;

VOID v_get_pixel        _((WORD handle, WORD x, WORD y, WORD FAR *pel,
                           WORD FAR *index));
VOID vro_cpyfm          _((WORD handle, WORD wr_mode, WORD FAR *xy,
                           MFDB FAR *srcMFDB, MFDB FAR *desMFDB));
VOID vr_trnfm           _((WORD handle, MFDB FAR *srcMFDB, MFDB FAR *desMFDB));
VOID vrt_cpyfm          _((WORD handle, WORD wr_mode, WORD FAR *xy,
                           MFDB FAR *srcMFDB, MFDB FAR *desMFDB,
                           WORD FAR *index));

/****** Input library ************************************************************

#define DEV_LOCATOR      1 /* input device */
#define DEV_VALUATOR     2
#define DEV_CHOICE       3
#define DEV_STRING       4
```

```
#define MODE_REQUEST     1 /* input mode */
#define MODE_SAMPLE      2

VOID vsin_mode          _((WORD handle, WORD dev_type, WORD mode));
VOID vrq_locator        _((WORD handle, WORD initx, WORD inity,
                           WORD FAR *xout, WORD FAR *yout, WORD FAR *term));
WORD vsm_locator        _((WORD handle, WORD initx, WORD inity,
                           WORD FAR *xout, WORD FAR *yout, WORD FAR *term));
VOID vrq_valuator       _((WORD handle, WORD val_in, WORD FAR *val_out,
                           WORD FAR *term));
VOID vsm_valuator       _((WORD handle, WORD val_in, WORD FAR *val_out,
                           WORD FAR *term, WORD FAR *status));
VOID vrq_choice         _((WORD handle, WORD in_choice, WORD FAR *out_choice));
WORD vsm_choice         _((WORD handle, WORD FAR *choice));
VOID vrq_string         _((WORD handle, WORD length, WORD echo_mode,
                           WORD FAR *echo_xy, BYTE FAR *string));
WORD vsm_string         _((WORD handle, WORD length, WORD echo_mode,
                           WORD FAR *echo_xy, BYTE FAR *string));
VOID vsc_form           _((WORD handle, WORD FAR *cur_form));
VOID vex_timv           _((WORD handle, WORD (FAR *tim_addr)(),
                           WORD (FAR * FAR *old_addr)(), WORD FAR *scale));
VOID v_show_c           _((WORD handle, WORD reset));
VOID v_hide_c           _((WORD handle));
VOID vq_mouse           _((WORD handle, WORD FAR *status, WORD FAR *px,
                           WORD FAR *py));
VOID vex_butv           _((WORD handle, WORD (FAR *usercode)(),
                           WORD (FAR * FAR *savcode)()));
VOID vex_motv           _((WORD handle, WORD (FAR *usercode)(),
                           WORD (FAR * FAR *savecode)()));
VOID vex_curv           _((WORD handle, WORD (FAR *usercode)(),
                           WORD (FAR * FAR *savecode)()));
VOID vq_key_s           _((WORD handle, WORD FAR *status));

/****** Inquire library ************************************************/

VOID vq_color           _((WORD handle, WORD index, WORD set_flag,
                           WORD FAR *rgb));
VOID vq_cellarray       _((WORD handle, WORD FAR *pxyarray,
                           WORD row_length, WORD num_rows,
                           WORD FAR *el_used, WORD FAR *rows_used,
                           WORD FAR *status, WORD FAR *colarray));
VOID vql_attributes     _((WORD handle, WORD FAR *attrib));
VOID vqm_attributes     _((WORD handle, WORD FAR *attrib));
VOID vqf_attributes     _((WORD handle, WORD FAR *attrib));
VOID vqt_attributes     _((WORD handle, WORD FAR *attrib));
VOID vq_extnd           _((WORD handle, WORD owflag, WORD FAR *work_out));
```

```
VOID vqin_mode          _((WORD handle, WORD dev_type, WORD FAR *input_mode));
VOID vqt_extent         _((WORD handle, BYTE FAR *string, WORD FAR *extent));
WORD vqt_width          _((WORD handle, BYTE character, WORD FAR *cell_width,
                           WORD FAR *left_delta, WORD FAR *right_delta));
WORD vqt_name           _((WORD handle, WORD element_num, BYTE FAR *name));
VOID vqt_font_info      _((WORD handle, WORD FAR *minADE, WORD FAR *maxADE,
                           WORD FAR *distances, WORD FAR *maxwidth,
                           WORD FAR *effects));
WORD vqt_justified      _((WORD handle, WORD x, WORD y, BYTE FAR *string,
                           WORD length, WORD word_space, WORD char_space,
                           WORD FAR *offsets));

/****** Escape library ***************************************************/

#define O_B_BOLDFACE     '0' /* OUT-File definitions for v_alpha_text */
#define O_E_BOLDFACE     '1'
#define O_B_ITALICS      '2'
#define O_E_ITALICS      '3'
#define O_B_UNDERSCORE   '4'
#define O_E_UNDERSCORE   '5'
#define O_B_SUPERSCRIPT  '6'
#define O_E_SUPERSCRIPT  '7'
#define O_B_SUBSCRIPT    '8'
#define O_E_SUBSCRIPT    '9'
#define O_B_NLQ          'A'
#define O_E_NLQ          'B'
#define O_B_EXPANDED     'C'
#define O_E_EXPANDED     'D'
#define O_B_LIGHT        'E'
#define O_E_LIGHT        'F'
#define O_PICA           'W'
#define O_ELITE          'X'
#define O_CONDENSED      'Y'
#define O_PROPORTIONAL   'Z'

#define O_GRAPHICS       "\033\033GEM,%d,%d,%d,%d,%s"

#define MUTE_RETURN     -1 /* definitions for vs_mute */
#define MUTE_ENABLE      0
#define MUTE_DISABLE     1

#define OR_PORTRAIT      0 /* definitions for v_orient */
#define OR_LANDSCAPE     1

#define TRAY_MANUAL     -1 /* definitions for v_tray */
#define TRAY_DEFAULT     0
#define TRAY_FIRSTOPT    1
```

```
#define XBIT_FRACT      0 /* definitions for v_xbit_image */
#define XBIT_INTEGER    1


#define XBIT_LEFT       0
#define XBIT_CENTER     1
#define XBIT_RIGHT      2


#define XBIT_TOP        0
#define XBIT_MIDDLE     1
#define XBIT_BOTTOM     2


VOID vq_chcells         _((WORD handle, WORD FAR *rows, WORD FAR *columns));
VOID v_exit_cur         _((WORD handle));
VOID v_enter_cur        _((WORD handle));
VOID v_curup            _((WORD handle));
VOID v_curdown          _((WORD handle));
VOID v_curright         _((WORD handle));
VOID v_curleft          _((WORD handle));
VOID v_curhome          _((WORD handle));
VOID v_eeos             _((WORD handle));
VOID v_eeol             _((WORD handle));
VOID vs_curaddress      _((WORD handle, WORD row, WORD column));
VOID v_curtext          _((WORD handle, BYTE FAR *string));
VOID v_rvon             _((WORD handle));
VOID v_rvoff            _((WORD handle));
VOID vq_curaddress      _((WORD handle, WORD FAR *row, WORD FAR *column));
WORD vq_tabstatus       _((WORD handle));
VOID v_hardcopy         _((WORD handle));
VOID v_dspcur           _((WORD handle, WORD x, WORD y));
VOID v_rmcur            _((WORD handle));
VOID v_form_adv         _((WORD handle));
VOID v_output_window    _((WORD handle, WORD FAR *xyarray));
VOID v_clear_disp_list  _((WORD handle));
VOID v_bit_image        _((WORD handle, CONST BYTE FAR *filename,
                           WORD aspect, WORD x_scale, WORD y_scale,
                           WORD h_align, WORD v_align, WORD FAR *xyarray));
VOID vq_scan            _((WORD handle, WORD FAR *g_height, WORD FAR *g_slices,
                           WORD FAR *a_height, WORD FAR *a_slices,
                           WORD FAR *factor));
VOID v_alpha_text       _((WORD handle, BYTE FAR *string));
WORD vs_palette         _((WORD handle, WORD palette));
VOID v_sound            _((WORD handle, WORD frequency, WORD duration));
WORD vs_mute            _((WORD handle, WORD action));
VOID vt_resolution      _((WORD handle, WORD xres, WORD yres,
                           WORD FAR *xset, WORD FAR *yset));
```

```
VOID vt_axis            _((WORD handle, WORD xres, WORD yres,
                           WORD FAR *xset, WORD FAR *yset));
VOID vt_origin          _((WORD handle, WORD xorigin, WORD yorigin));
VOID vq_tdimensions     _((WORD handle, WORD FAR *xdimension,
                           WORD FAR *ydimension));
VOID vt_alignment       _((WORD handle, WORD dx, WORD dy));
VOID vsp_film           _((WORD handle, WORD index, WORD lightness));
WORD vqp_filmname       _((WORD handle, WORD index, BYTE FAR *name));
VOID vsc_expose         _((WORD handle, WORD state));
VOID v_meta_extents     _((WORD handle, WORD min_x, WORD min_y,
                           WORD max_x, WORD max_y));
VOID v_write_meta       _((WORD handle, WORD num_intin, WORD FAR *intin,
                           WORD num_ptsin, WORD FAR *ptsin));
VOID vm_filename        _((WORD handle, CONST BYTE FAR *filename));
VOID vm_pagesize        _((WORD handle, WORD pgwidth, WORD pgheight));
VOID vm_coords          _((WORD handle, WORD llx, WORD lly,
                           WORD urx, WORD ury));
VOID v_copies           _((WORD handle, WORD count));
VOID v_orient           _((WORD handle, WORD orientation));
VOID v_tray             _((WORD handle, WORD tray));
WORD v_xbit_image       _((WORD handle, BYTE FAR *filename, WORD aspect,
                           WORD x_scale, WORD y_scale,
                           WORD h_align, WORD v_align, WORD rotate,
                           WORD background, WORD foreground, WORD FAR *xy));
VOID vs_bkcolor         _((WORD handle, WORD color));
VOID v_setrgbi          _((WORD handle, WORD primtype, WORD r, WORD g, WORD b,
                           WORD i));
VOID v_topbot           _((WORD handle, WORD height,
                           WORD FAR *char_width, WORD FAR *char_height,
                           WORD FAR *cell_width, WORD FAR *cell_height));
VOID v_ps_halftone      _((WORD handle, WORD index, WORD angle,
                           WORD frequency));


#if GEM & GEM1
VOID vqp_films          _((WORD handle, BYTE FAR *film_names));
VOID vqp_state          _((WORD handle, WORD FAR *port, BYTE FAR *film_name,
                           WORD FAR *lightness, WORD FAR *interlace,
                           WORD FAR *planes, WORD FAR *indexes));
VOID vsp_state          _((WORD handle, WORD port, WORD film_num,
                           WORD lightness, WORD interlace, WORD planes,
                           WORD FAR *indexes));
VOID vsp_save           _((WORD handle));
VOID vsp_message        _((WORD handle));
WORD vqp_error          _((WORD handle));
```

```
VOID v_offset           _((WORD handle, WORD offset));
VOID v_fontinit         _((WORD handle, WORD fh_high, WORD fh_low));
VOID v_escape2000       _((WORD times));
WORD vq_gdos            _((VOID));
#endif /* GEM1 */

/*****************************************************************************/

#endif /* __VDI__ */
```

# Anhang B. Make- und Projekt-Dateien

a) GEMDOS

- Digital Research C (Alcyon C)

Datei C.BAT:

```
cp68 %1.c %1.i
c068 %1.i %1.1 %1.2 %1.3 -f
del %1.i
c168 %1.1 %1.2 %1.s
del %1.1
del %1.2
as68 -l -u %1.s
del %1.s
ar68 rv scraplib %1.o
del %1.o
```

Datei C_ALL.BAT:

```
c gemain
c initerm
c graf
c power
c edit
c image
c meta
c clipbrd
c printer
c disk
c trash
c desktop
```

```
c event
c menu
c resource
c rcm
c windows
c global
l
```

Datei L.BAT:

```
link68 scrap.68k=gemstart,scraplib,vdibind,aesbind,osbind,gemlib,libf
relmod scrap.68k
del scrap.68k
```

- Laser C

Datei MAKEFILE:

```
##### MAKEFILE für SCRAP #####

#############################################################################

NAME   = scrap

#############################################################################

CFLAGS =

LFLAGS =

APP    = .prg

#############################################################################

H      = import.h export.h global.h

OBJS   = clipbrd.o desktop.o disk.o edit.o event.o gemain.o global.o graf.o\
         image.o initerm.o menu.o meta.o power.o printer.o rcm.o resource.o\
         trash.o windows.o

#############################################################################
```

```
$(NAME)$(APP): $(OBJS)
    $(CC) $(OBJS) $(LFLAGS) -o $@

##############################################################################

clipbrd.o : $(H) windows.h $(NAME).h resource.h desktop.h edit.h image.h\
            meta.h trash.h clipbrd.h
desktop.o : $(H) windows.h $(NAME).h clipbrd.h disk.h event.h menu.h printer.h\
            trash.h desktop.h
disk.o    : $(H) windows.h $(NAME).h disk.h
edit.o    : $(H) windows.h $(NAME).h resource.h desktop.h clipbrd.h edit.h
event.o   : $(H) windows.h desktop.h menu.h event.h
gemain.o  : $(H) windows.h initerm.h event.h gemain.h
global.o  : $(H)
graf.o    : $(H) windows.h $(NAME).h graf.h
image.o   : $(H) windows.h $(NAME).h resource.h clipbrd.h image.h
initerm.o : $(H) windows.h $(NAME).h resource.h menu.h event.h desktop.h\
            trash.h disk.h printer.h edit.h clipbrd.h image.h meta.h graf.h\
            power.h initerm.h
menu.o    : $(H) windows.h $(NAME).h desktop.h graf.h power.h resource.h menu.h
meta.o    : $(H) windows.h $(NAME).h resource.h clipbrd.h meta.h
power.o   : $(H) windows.h $(NAME).h power.h
printer.o : $(H) windows.h $(NAME).h printer.h
rcm.o     : rcm.h
resource.o: $(H) $(NAME).h resource.h
trash.o   : $(H) windows.h $(NAME).h desktop.h clipbrd.h trash.h
windows.o : $(H) windows.h

##############################################################################
```

- Lattice C

Datei SCRAP.LNK:

```
*
* Standard control file for linking Lattice C modules.
*
* Step 1 - initialisation
* =======================
*
* C initialisation must be included first.
*
INPUT   startup.bin
*
* Step 2 - user modules
* =====================
*
* Now include a single user module
* (from the command line).
*
* INPUT *
INPUT clipbrd.bin
INPUT desktop.bin
INPUT disk.bin
INPUT edit.bin
INPUT event.bin
INPUT gemain.bin
INPUT global.bin
INPUT graf.bin
INPUT image.bin
INPUT initerm.bin
INPUT menu.bin
INPUT meta.bin
INPUT power.bin
INPUT printer.bin
INPUT rcm.bin
INPUT resource.bin
INPUT trash.bin
INPUT windows.bin
*
* For each extra module you want to include in the
* link include a line of the form:
*
* INPUT <file name>
*
* Step 3 - floating point library
```

```
* ================================

* Comment out this line if your program does not use
* floating point
*
LIBRARY fplib.bin
*
* Step 4 - C library
* ==================
*
* C library - must always be included.
*
LIBRARY clib.bin
*
* Step 5 - GEM graphics library
* ============================
*
* GEM graphics library - only include if your program
* is trying to access graphics routines
* (by uncommenting the line).
*
LIBRARY gemlib.bin
*
```

- Mark Williams C

Datei MAKEFILE:

```
#### MAKEFILE für SCRAP ####

##############################################################################

NAME   = scrap

##############################################################################

CFLAGS = -VPEEP

LFLAGS = -x -s -laes -lvdi

APP    = .prg

##############################################################################
```

```
H       = import.h export.h global.h


OBJS    = clipbrd.o desktop.o disk.o edit.o event.o gemain.o global.o graf.o\
          image.o initerm.o menu.o meta.o power.o printer.o rcm.o resource.o\
          trash.o windows.o

#############################################################################

$(NAME)$(APP): $(OBJS)
    $(CC) $(OBJS) $(LFLAGS) -o $@

#############################################################################

clipbrd.o : $(H) windows.h $(NAME).h resource.h desktop.h edit.h image.h\
            meta.h trash.h clipbrd.h
desktop.o : $(H) windows.h $(NAME).h clipbrd.h disk.h event.h menu.h printer.h
            trash.h desktop.h
disk.o    : $(H) windows.h $(NAME).h disk.h
edit.o    : $(H) windows.h $(NAME).h resource.h desktop.h clipbrd.h edit.h
event.o   : $(H) windows.h desktop.h menu.h event.h
gemain.o  : $(H) windows.h initerm.h event.h gemain.h
global.o  : $(H)
graf.o    : $(H) windows.h $(NAME).h graf.h
image.o   : $(H) windows.h $(NAME).h resource.h clipbrd.h image.h
initerm.o : $(H) windows.h $(NAME).h resource.h menu.h event.h desktop.h\
            trash.h disk.h printer.h edit.h clipbrd.h image.h meta.h graf.h\
            power.h initerm.h
menu.o    : $(H) windows.h $(NAME).h desktop.h graf.h power.h resource.h menu.
meta.o    : $(H) windows.h $(NAME).h resource.h clipbrd.h meta.h
power.o   : $(H) windows.h $(NAME).h power.h
printer.o : $(H) windows.h $(NAME).h printer.h
rcm.o     : rcm.h
resource.o: $(H) $(NAME).h resource.h
trash.o   : $(H) windows.h $(NAME).h desktop.h clipbrd.h trash.h
windows.o : $(H) windows.h

#############################################################################
```

- Turbo C

Datei SCRAP.PRJ:

```
                ; SCRAP.PRJ
                ; ----------
.C[-W-par -W-sig]
=               ; list of modules follows...
tcstart.o       ; startup code
clipbrd         (global.h, windows.h, scrap.h, resource.h, desktop.h, edit.h,
                 image.h, meta.h, trash.h, clipbrd.h)
desktop         (global.h, windows.h, scrap.h, clipbrd.h, disk.h, event.h,
                 menu.h, printer.h, trash.h, desktop.h)
disk            (global.h, windows.h, scrap.h, disk.h)
edit            (global.h, windows.h, scrap.h, resource.h, desktop.h, clipbrd.h,
                 edit.h)
event           (global.h, windows.h, desktop.h, menu.h, event.h)
gemain          (global.h, windows.h, initerm.h, event.h, gemain.h)
global          (global.h)
graf            (global.h, windows.h, scrap.h, graf.h)
image           (global.h, windows.h, scrap.h, resource.h, clipbrd.h, image.h)
initerm         (global.h, windows.h, scrap.h, resource.h, menu.h, event.h,
                 desktop.h, trash.h, disk.h, printer.h, edit.h, clipbrd.h,
                 image.h, meta.h, graf.h, power.h, initerm.h)
menu            (global.h, windows.h, scrap.h, desktop.h, graf.h, power.h,
                 resource.h, menu.h)
meta            (global.h, windows.h, scrap.h, resource.h, clipbrd.h, meta.h)
power           (global.h, windows.h, scrap.h, power.h)
printer         (global.h, windows.h, scrap.h, printer.h)
rcm             (rcm.h)
resource        (global.h, scrap.h, resource.h) [-W-rpt]
trash           (global.h, windows.h, scrap.h, desktop.h, clipbrd.h, trash.h)
windows         (global.h, windows.h)
tcfltlib.lib    ; floating point lib
tcstdlib.lib    ; standard lib
tcextlib.lib    ; extended lib
tctoslib.lib    ; TOS lib
tcgemlib.lib    ; AES and VDI lib
```

b) MSDOS

- Microsoft C

Datei MK.BAT:

```
make makefile
```

Datei MAKEFILE:

```
#############################################################################

NAME   = scrap

#############################################################################

CFLAGS = -AL -Oas -Gs

LFLAGS = /ST:12288 /SE:512 /NOI /E

APP    = .app

#############################################################################

.c.obj:
    cl -c $(CFLAGS) $*.c

.asm.obj:
    masm /Mx $*;

#############################################################################

H      = $*.c $*.h import.h export.h global.h

OBJS   = clipbrd.obj desktop.obj disk.obj edit.obj event.obj gemain.obj\
         global.obj graf.obj image.obj initerm.obj menu.obj meta.obj\
         power.obj printer.obj rcm.obj resource.obj trash.obj windows.obj

#############################################################################

clipbrd.obj : $(H) windows.h $(NAME).h desktop.h edit.h image.h meta.h trash.h

desktop.obj : $(H) windows.h $(NAME).h disk.h event.h menu.h printer.h trash.h
              clipbrd.h
```

```
disk.obj     : $(H) windows.h $(NAME).h

edit.obj     : $(H) windows.h $(NAME).h resource.h desktop.h clipbrd.h

event.obj    : $(H) windows.h desktop.h menu.h

gemain.obj   : $(H) initerm.h event.h

global.obj   : $(H)

graf.obj     : $(H) windows.h $(NAME).h

image.obj    : $(H) windows.h $(NAME).h resource.h clipbrd.h

initerm.obj  : $(H) windows.h $(NAME).h resource.h menu.h event.h desktop.h\
               trash.h disk.h printer.h clipbrd.h edit.h image.h meta.h graf.h\
               power.h

menu.obj     : $(H) windows.h $(NAME).h desktop.h graf.h power.h resource.h

meta.obj     : $(H) windows.h $(NAME).h resource.h clipbrd.h

power.obj    : $(H) windows.h $(NAME).h resource.h

printer.obj  : $(H) windows.h $(NAME).h

rcm.obj      : $(H)

resource.obj: $(H) $(NAME).h

trash.obj    : $(H) windows.h $(NAME).h desktop.h clipbrd.h

windows.obj  : $(H)

#############################################################################

$(NAME)$(APP) : $(OBJS)
    link $(LFLAGS) @$(NAME).lnk

#############################################################################
```

Datei SCRAP.LNK:

```
clipbrd+
desktop+
disk+
edit+
event+
gemain+
global+
graf+
image+
initerm+
menu+
meta+
power+
printer+
rcm+
resource+
trash+
windows
scrap.app

lmscgem
```

- Turbo C

Datei BUILTINS.MAK:

```
.c.obj:
    tcc -c $(CFLAGS) $*

.asm.obj:
    tasm /mx $*
```

Datei MAKEFILE:

```
##### MAKEFILE für SCRAP #####

#############################################################################

NAME   = scrap

#############################################################################
```

```
CFLAGS = -ml -w-par

LFLAGS = /c /x

APP    = .app

#############################################################################

H      = import.h export.h global.h

OBJS   = clipbrd.obj desktop.obj disk.obj edit.obj event.obj gemain.obj\
         global.obj graf.obj image.obj initerm.obj menu.obj meta.obj\
         power.obj printer.obj rcm.obj resource.obj trash.obj windows.obj

#############################################################################

$(NAME)$(APP): $(OBJS)
    tlink $(LFLAGS) @$(NAME).lnk

#############################################################################

clipbrd.obj : $(H) windows.h $(NAME).h resource.h desktop.h edit.h image.h\
              meta.h trash.h clipbrd.h
desktop.obj : $(H) windows.h $(NAME).h clipbrd.h disk.h event.h menu.h\
              printer.h trash.h desktop.h
disk.obj    : $(H) windows.h $(NAME).h disk.h
edit.obj    : $(H) windows.h $(NAME).h resource.h desktop.h clipbrd.h edit.h
event.obj   : $(H) windows.h desktop.h menu.h event.h
gemain.obj  : $(H) initerm.h event.h gemain.h
global.obj  : $(H)
graf.obj    : $(H) windows.h $(NAME).h graf.h
image.obj   : $(H) windows.h $(NAME).h resource.h clipbrd.h image.h
initerm.obj : $(H) windows.h $(NAME).h resource.h menu.h event.h desktop.h\
              trash.h disk.h printer.h edit.h clipbrd.h image.h meta.h graf.h\
              power.h initerm.h
menu.obj    : $(H) windows.h $(NAME).h desktop.h graf.h power.h resource.h\
              menu.h
meta.obj    : $(H) windows.h $(NAME).h resource.h clipbrd.h meta.h
power.obj   : $(H) windows.h $(NAME).h power.h
printer.obj : $(H) windows.h $(NAME).h printer.h
rcm.obj     : rcm.h
resource.obj: $(H) $(NAME).h resource.h
trash.obj   : $(H) windows.h $(NAME).h desktop.h clipbrd.h trash.h
windows.obj : $(H) windows.h

#############################################################################
```

Datei SCRAP.LNK:

```
c:\tc\lib\c01+
clipbrd+
desktop+
disk+
edit+
event+
gemain+
global+
graf+
image+
initerm+
menu+
meta+
power+
printer+
rcm+
resource+
trash+
windows
scrap.app

c:\tc\lib\ltcgem+c:\tc\lib\mathl+c:\tc\lib\emu+c:\tc\lib\cl
```

Datei SCRAP.PRJ:

```
clipbrd      (global.h, windows.h, scrap.h, resource.h, desktop.h, edit.h,
              image.h, meta.h, trash.h, clipbrd.h)
desktop      (global.h, windows.h, scrap.h, clipbrd.h, disk.h, event.h,
              menu.h, printer.h, trash.h, desktop.h)
disk         (global.h, windows.h, scrap.h, disk.h)
edit         (global.h, windows.h, scrap.h, resource.h, desktop.h, clipbrd.h
              edit.h)
event        (global.h, windows.h, desktop.h, menu.h, event.h)
gemain       (global.h, windows.h, initerm.h, event.h, gemain.h)
global       (global.h)
graf         (global.h, windows.h, scrap.h, graf.h)
image        (global.h, windows.h, scrap.h, resource.h, clipbrd.h, image.h)
initerm      (global.h, windows.h, scrap.h, resource.h, menu.h, event.h,
              desktop.h, trash.h, disk.h, printer.h, edit.h, clipbrd.h,
              image.h, meta.h, graf.h, power.h, initerm.h)
menu         (global.h, windows.h, scrap.h, desktop.h, graf.h, power.h,
              resource.h, menu.h)
meta         (global.h, windows.h, scrap.h, resource.h, clipbrd.h, meta.h)
power        (global.h, windows.h, scrap.h, power.h)
```

```
printer         (global.h, windows.h, scrap.h, printer.h)
rcm             (rcm.h)
resource        (global.h, scrap.h, resource.h)
trash           (global.h, windows.h, scrap.h, desktop.h, clipbrd.h, trash.h)
windows         (global.h, windows.h)
ltcgem.lib
```

Datei TURBOC.CFG:

```
-ml
-Ic:\tc\include
-Lc:\tc\lib
```

c) FlexOS

- High C

Datei MAKEFILE:

```
##### MAKEFILE für SCRAP #####

#############################################################################

NAME   = scrap

#############################################################################

CFLAGS = -Off Prototype_overwrite_warnings -On PCC_msgs

LFLAGS =

APP    = .286

#############################################################################

COMPILE = hc $*.c $(CFLAGS)

#############################################################################

H      = import.h export.h global.h

OBJS   = clipbrd.obj desktop.obj disk.obj edit.obj event.obj gemain.obj\
         global.obj graf.obj image.obj initerm.obj menu.obj meta.obj\
         power.obj printer.obj rcm.obj resource.obj trash.obj windows.obj
```

```
#############################################################################

$(NAME)$(APP): $(OBJS)
    link86 $(LFLAGS) $(NAME).inp[i]

#############################################################################

clipbrd.obj : clipbrd.c clipbrd.h windows.h $(NAME).h resource.h desktop.h\
              edit.h image.h meta.h trash.h
    $(COMPILE)

desktop.obj : desktop.c desktop.h windows.h $(NAME).h disk.h event.h menu.h\
              printer.h trash.h clipbrd.h
    $(COMPILE)

disk.obj    : disk.c disk.h windows.h $(NAME).h
    $(COMPILE)

edit.obj    : edit.c edit.h windows.h $(NAME).h resource.h desktop.h clipbrd.h
    $(COMPILE)

event.obj   : event.c event.h windows.h desktop.h menu.h
    $(COMPILE)

gemain.obj  : gemain.c gemain.h initerm.h event.h
    $(COMPILE)

global.obj  : global.c global.h
    $(COMPILE)

graf.obj    : graf.c graf.h windows.h $(NAME).h
    $(COMPILE)

image.obj   : image.c image.h windows.h $(NAME).h resource.h clipbrd.h
    $(COMPILE)

initerm.obj : initerm.c initerm.h windows.h $(NAME).h resource.h menu.h\
              event.h desktop.h trash.h disk.h printer.h clipbrd.h edit.h\
              image.h meta.h graf.h power.h
    $(COMPILE)


menu.obj    : menu.c menu.h windows.h $(NAME).h desktop.h graf.h power.h\
              resource.h
    $(COMPILE)

meta.obj    : meta.c meta.h windows.h $(NAME).h resource.h clipbrd.h
    $(COMPILE)

power.obj   : power.c power.h windows.h $(NAME).h resource.h
    $(COMPILE)
```

```
printer.obj : printer.c printer.h windows.h $(NAME).h
    $(COMPILE)

rcm.obj     : rcm.c rcm.h
    $(COMPILE)

resource.obj: resource.c resource.h $(NAME).h
    $(COMPILE)

trash.obj   : trash.c trash.h windows.h $(NAME).h desktop.h clipbrd.h
    $(COMPILE)

windows.obj : windows.c windows.h
    $(COMPILE)
```

```
##############################################################################
```

Datei SCRAP.INP

```
scrap[stack[add[4000]]] =
xgembig.186[nose,li],
clipbrd,
desktop,
disk,
edit,
event,
gemain,
global,
graf,
image,
initerm,
menu,
meta,
power,
printer,
rcm,
resource,
trash,
windows,
xgemsrtl.186[s,li],
dosrtl.186[s,li],
hcbe.186[s,li]
```

Datei SCRAP.PRF

```
Graphics Mode = True
```

# Anhang C. Die neue Software

Seit der Erstauflage dieses Buches ist nunmehr etwa ein Jahr verstrichen. Das Demoprogramm zum Buch (SCRAP) wurde ständig weiterentwickelt und liegt nun in einer überarbeiteten Version vor. Zunächst einige Schlagworte zu den wesentlichen Änderungen.

- Unterstützung beider Mausknöpfe
- modale und nicht-modale Dialogboxen in echten Fenstern
- erweiterte Alertboxen in echten Fenstern
- blinde Paßwort-Eingabe
- Unterstützung der TOS 1.4 File-Selector-Box
- Listboxen in Dialogboxen

Außerdem wurden viele Kleinigkeiten verbessert, von denen die meisten durch die folgenden Erläuterungen klar werden dürften.

Die Compiler Digital Research C und Lattice-C werden nicht mehr unterstützt, d.h. nicht mehr getestet. Dies bedeutet nicht, daß das Programm SCRAP nicht auch mit diesen Compilern laufen würde. Beim Digital Research Compiler würden sich allerdings Probleme ergeben, die darauf zurückzuführen sind, daß die Länge der Bezeichner auf 7 (für externe Bezeichner) beschränkt ist.

Leider mußten wir auch feststellen, daß die modalen und nicht-modalen Fenster mit dem Megamax Laser C Compiler nicht richtig funktionieren, wenn sie editierbare Felder beinhalten. Offensichtlich sind die dafür notwendigen und selten benutzten GEM-Bindings "objc_edit", "form_keybd" und "form_button" nicht richtig implementiert. Arbeitet man mit diesem Compiler, muß man also auf die genannten Features verzichten. Da aber auch die benutzerdefinierbaren Objekte mit diesem Compiler nicht laufen können, ist ein Umsteigen auf einen anderen Compiler sinnvoll.

Der Turbo C Compiler wird ab der Version 2.0 unterstützt. Es soll noch erwähnt werden, daß das Stack Checking ausgeschaltet sein muß (wegen der benutzerdefinierten Objekte) und absolute Calls benutzt werden müssen. Außerdem sollten Zeichenketten nicht zusammengelegt werden, also

–T aus
–P an
–M an

Im folgenden wird erläutert, welche wesentlichen Änderungen die einzelnen Module erfahren haben.

# Modul GLOBAL

Folgende Konstanten-Definitionen wurden aufgenommen:

```
#define FONT_SYSTEM     1
#define FONT_SWISS      2
#define FONT_DUTCH      14
```

Es handelt sich um den System-Zeichensatz, sowie die beiden Zeichensätze, die von Digital Research zum GEM/3 mitgeliefert werden. Die Konstanten können bei einem Aufruf von "vst_font" benutzt werden.

```
#define POS1            0x47
#define PGUP            0x49
#define ENDKEY          0x4F
#define PGDOWN          0x51
#define CNTRL_LEFT      0x73
#define CNTRL_RIGHT     0x74
```

Die ersten vier Definitionen bezeichnen die Scan-Codes der Tasten eines AT-Keyboards, die beim Atari ST nicht vorhanden sind. Die beiden letzten Definitionen wurden aufgenommen, da die Pfeiltasten mit gedrückter Control-Taste einen anderen Scan-Code liefern.

```
#define UNDO_FLAG       0x0200
#define HELP_FLAG       0x0400
#define NOECHO_FLAG     0x0800
```

Die drei Flags werden zusätzlich zu den OB_FLAGS eines Objektes benutzt. Da sie nicht interaktiv im Resource-Construction-Set eingegeben werden können, werden sie im Module RESOURCE "zu Fuß" gesetzt (siehe Modul RESOURCE).

Das UNDO-Flag gibt an, welcher Knopf (Button) einer Dialogbox mit der UNDO-Taste aufgerufen wird, falls diese Taste vorhanden ist.

Das HELP-Flag gibt an, welcher Knopf (Button) einer Dialogbox mit der HELP-Taste aufgerufen wird, falls diese Taste vorhanden ist. Im Programm wird die Funktionstaste F1 zur zusätzlichen Help-Taste.

Das NOECHO-Flag gibt an, welches editierbare Objekt einer Dialogbox kein Echo haben soll. In ein solches Objekt werden dann anstatt Buchstaben '*'-Symbole ausgegeben. Dies dient z.B. zur Eingabe von Paßwörtern. Ein Beispiel wird im Modul MENU gegeben. Die dort zu öffnende Dialogbox für die Systemeinstellungen beinhaltet auch ein Paßwort. Weitere Informationen wird in der Beschreibung zur Funktion "edit_noecho" gegeben.

```
typedef BYTE HUGE *HPTR;
typedef BYTE FAR  *FPTR;
```

Die beiden Typdefinitionen sind unter MS-DOS zu verwenden. Dort stellen sie Huge-Pointer bzw. Far-Pointer dar.

```
typedef struct
{
...
  WORD momask;
...
} MKINFO;
```

In der Struktur MKINFO (Mouse-Keyboard-Info) ist ein Element dazugekommen. Es handelt sich um die Komponente "momask". Sie gibt die Maske des Mausknopfes an, der ein Ereignis ausgelöst hat. Früher wurde nur die Komponente "mobutton" zu Hilfe genommen. Da nun zwei Mausknöpfe unterstützt werden, benötigt man zusätzliche Information, da man aus der Belegung der Komponenten "mobutton" mit der Zahl 0 (bei Mausknopf oben) nicht erkennen kann, welcher Mausknopf nun oben ist.

"momask" kann folgende Werte annehmen:

0x0001 : linker Mausknopf hat Ereignis ausgelöst
0x0002 : rechter Mausknopf hat Ereignis ausgelöst

"mobutton" nimmt nach wie vor folgende Werte an:

0x0000 : Mausknöpfe oben
0x0001 : linker Mausknopf unten
0x0002 : rechter Mausknopf unten

Damit ergeben sich zur Unterscheidung der einzelnen Ereignisse folgende Werte:

| Ereignis | momask | mobutton |
|---|---|---|
| Klick links | 0x0001 | 0x0000 |
| Drag links | 0x0001 | 0x0001 |
| Klick rechts | 0x0002 | 0x0000 |
| Drag rechts | 0x0002 | 0x0002 |

Klick bedeutet, daß der Mausknopf sofort wieder losgelassen wird, Drag, daß der Mausknopf gedrückt gehalten wird. Wird ein Mausknopf gedrückt, wird vom Modul EVENT auf ein Loslassen desselben gewartet, so daß ein durchgehendes Drücken nur ein Ereignis auslöst. Soll dies verhindert werden, so muß in der Routine, die dieses Ereignis verarbeitet, der Wert der Variablen "bclicks" auf 0x102 gesetzt werden (siehe auch Modul DESKTOP, Routine "wi_click").

GLOBAL WORD gl_point;

Die Punktgröße des Systemzeichensatzes. Sie kann z.B. in einem Aufruf "vst_point" benutzt werden. Im allgemeinen hat sie den Wert 10. Auf dem Atari ST in mittlerer oder niedriger Auflösung hat sie beispielsweise den Wert 9.

GLOBAL WORD mousenumber;

Die aktuelle Nummer der Mausform. Wird z.B. im Modul DIALOG benutzt, um nach der Behandlung einer modalen Dialogbox die ursprüngliche Mausform wiederherzustellen.

GLOBAL MFORM *mouseform;

Die aktuelle Form der Maus, falls "mousenumber" den Wert USER_DEF hat.

GLOBAL BOOLEAN updtmenu;

Flag, welches angibt, ob nach einem Ereignis die Menüs auf den neuesten Stand gebracht werden sollen. Normalerweise ist dies der Fall, da nach einem Ereignis meist ein Zustand eintritt, der Menüs verändert bzw. Menüeinträge ein- oder ausschaltet. Wird die Variable von einer Routine auf FALSE gesetzt, so wird beim nächsten Durchgang das Menü nicht auf den neuesten Stand gebracht.

Der Sinn, der dahinter steckt ist einfach der, Zeit zu sparen, da die Funktion "updt_menu" oft aufgerufen wird und etwas Zeit kostet. Im Normalfall macht sich dies aber nicht bemerkbar.

Einige Funktionen des Moduls WINDOWS setzen die Variable auf FALSE, da sich kein Menü ändern kann. So wird dies z.B. beim Verschieben eines Fensters getan oder wenn eine Taste in einem editierbaren Objekt in einer Dialogbox gedrückt wird, die sich in einem Fenster befindet. Dabei ändert sich der Zustand des Programmes im Normalfall nicht.

GLOBAL BYTE **alerts;

Die Variable enthält eine Tabelle von Zeigern auf die Fehlermeldungen. Fehlermeldungen werden nicht mehr im Resource-Construction-Set angegeben, da sie dann nicht flexibel genug sind. Die Fehlerbehandlung wird noch weiter unten erklärt.

BOOLEAN is_state (OBJECT *tree, WORD obj, UWORD state);

Es wird geprüft, ob ein bestimmter Status (z.B. SELECTED) eines Objektes (ob_state) gesetzt ist.

is_state : TRUE, wenn der geprüfte Status gesetzt ist, sonst FALSE
tree : Zeiger auf einen Objektbaum
obj : Index des Objekts
state : Status, der geprüft werden soll

BOOLEAN is_flags (OBJECT *tree, WORD obj, UWORD flags);

Es wird geprüft, ob ein bestimmtes Flag (z.B. SELECTABLE) eines Objektes (ob_flags) gesetzt ist.

is_flags : TRUE, wenn das geprüfte Flag gesetzt ist, sonst FALSE
tree : Zeiger auf einen Objektbaum
obj : Index des Objekts
flags : Flag, welches geprüft werden soll

WORD find_type (OBJECT *tree, WORD obj, UWORD type);

Sucht ein Objekt eines bestimmten Typs in einem Objektbaum.

find_type : Index des gesuchten Objekts im Objektbaum oder NIL, wenn kein solches Objekt gefunden wurde

tree : Zeiger auf einen Objektbaum
obj : Index des ersten zu untersuchenden Objekts
type : Typ, den das gesuchte Objekt haben soll

VOID set_checkbox (OBJECT *tree, WORD obj, BOOLEAN selected);

Der Status SELECTED eines Objekts (Checkbox) wird gesetzt oder zurückgesetzt.

tree : Zeiger auf einen Objektbaum
obj : Index des Objekts
selected : Ist der Wert TRUE, wird der Status SELECTED gesetzt, sonst zurückgesetzt

BOOLEAN get_checkbox (OBJECT *tree, WORD obj);

Liefert den Status eines Objekts (Checkbox).

get_checkbox : TRUE, wenn der Status SELECTED gesetzt ist, sonst FALSE
tree : Zeiger auf einen Objektbaum
obj : Index des Objekts

VOID set_rbutton (OBJECT *tree, WORD obj, WORD lower, WORD upper);

Der Status SELECTED einer Gruppe von Objekten (Radio-Buttons) wird gesetzt oder zurückgesetzt. Dabei wird ein Objekt selektiert, alle anderen deselektiert.

tree : Zeiger auf einen Objektbaum
obj : Index des Objekts, welches selektiert werden soll
lower : Index des ersten Objekts (Radio-Buttons)
upper : Index des letzten Objekts (Radio-Buttons)

WORD get_rbutton (OBJECT *tree, WORD obj);

Die Objektnummer eines selektierten Objekts (Radio-Buttons) einer Gruppe von Objekten wird geliefert. Bei Aufruf der Funktion muß ein Objekt selektiert sein.

get_rbutton : Index des ersten gefundenen selektierten Objekts
tree : Zeiger auf einen Objektbaum
obj : Index des ersten Objekts, ab dem gesucht werden soll

VOID menu_check (OBJECT *tree, WORD obj, BOOLEAN checkit);

Es handelt sich um die gleiche Funktion wie "menu_icheck", welche vom GEM bekannt ist. Der Unterschied besteht nur in der wesentlich schnelleren Ausführung von "menu_check".

VOID menu_enable (OBJECT *tree, WORD obj, BOOLEAN enableit);

Es handelt sich um die gleiche Funktion wie "menu_ienable", welche vom GEM bekannt ist. Der Unterschied besteht nur in der wesentlich schnelleren Ausführung von "menu_enable". Dies macht sich vor allem in der Funktion "updt_menu" bemerkbar, die viele solche Aufrufe tätigt.

VOID objc_rect (OBJECT *tree, WORD obj, RECT *rect,
BOOLEAN calc_border);

Die aktuellen Koordinaten des Rechtecks, das ein Objekt umhüllt. Das Ergebnis bezieht sich auf den Koordinatenursprung in der linken oberen Ecke des Bildschirms (x = 0, y = 0). Es kann außerdem angegeben werden, ob der Rand bei Objekten des Typs G_BOX, G_IBOX, G_BOXCHAR mit berücksichtigt werden soll. Zu diesem Rand gehören auch SHADOWED und OUTLINED.

| | |
|---|---|
| tree | : Zeiger auf einen Objektbaum |
| obj | : Index des Objekts |
| rect | : Zeiger auf das resultierende Rechteck |
| calc_border | : TRUE, wenn der Rand mit berücksichtigt werden soll, sonst FALSE |

VOID line_default (WORD vdi_handle);

Setzt Zeichenmodus für Linien auf einen Standardwert. Dieser ist gegeben durch eine schwarze, durchgezogene Linie mit quadratischen Ecken der Stärke eins, welche im Replace-Modus gezeichnet wird.

vdi_handle : Handle der VDI-Arbeitsstation

VOID text_default (WORD vdi_handle);

Setzt Zeichenmodus für Text auf einen Standardwert. Dieser ist gegeben durch normale schwarze Buchstaben im System-Zeichensatz, die von links nach rechts im Replace-Modus gezeichnet werden.

vdi_handle : Handle der VDI-Arbeitsstation

VOID v_text (WORD vdi_handle, WORD x, WORD y,
BYTE *string, WORD charwidth);

Die Funktion ähnelt der VDI-Funktion "v_gtext", nur daß damit auch Texte eines nichtproportionalen Zeichensatzes ausgegeben werden können, welche länger als 127 Zeichen sein können. Dabei muß allerdings die Breite eines Zeichens mit angegeben werden.

| | |
|---|---|
| vdi_handle | : Handle der VDI-Arbeitsstation |
| x | : X-Koordinate des auszugebenden Textes |

| | |
|---|---|
| y | : Y-Koordinate des auszugebenden Textes |
| string | : Zeiger auf den auszugebenden Text |
| charwidth | : Breite eines Zeichens (z.B. gl_wbox) |

Die Funktion "open_dial" wurde in "opendial" umbenannt, da es eine neue Funktion "open_dialog" gibt.

Die Funktion "close_dial" wurde in "closedial" umbenannt.

Die Funktion "mem_free" liefert kein Ergebnis mehr zurück, da dies nicht in allen Systemumgebungen garantiert wird.

```
BOOLEAN select_file (BYTE *name, BYTE *path, BYTE *suffix,
                     BYTE *label, BYTE *filename);
```

Mit Hilfe der Dateiauswahl-Box von GEM kann eine Datei gewählt werden. Dabei kann ein Standard-Name (z.B. NAMENLOS.DOC) sowie ein Suffix angegeben werden (z.B. "*.DOC"). Ab TOS 1.4 kann auch ein Label angegeben werden. Unterstützt die GEM-Version dies nicht, wird das Label ignoriert.

| | |
|---|---|
| select_file | : TRUE, wenn eine Datei ausgewählt und OK gewählt wurde, sonst FALSE |
| name | : Zeiger auf einen Dateinamen der als Default-Namen angeboten wird |
| suffix | : Zeiger auf ein Suffix, welches als Default-Suffix angeboten wird |
| label | : Zeiger auf eine Zeichenkette, die als Beschriftung dient |
| filename | : Resultierender kompletter Dateiname |

Bei dieser Funktion werden auch die globalen Variablen "fs_path", "fs_sel" und "fs_button" gesetzt. Ist der Parameter "suffix" nicht leer, so wird er in die Dateiauswahl-Box gesetzt, ansonsten wird das zuletzt eingestellte Suffix benutzt.

Ist der Parameter "name" nicht leer, so wird er in die Dateiauswahl-Box gesetzt, ansonsten wird der zuletzt gewählte Name nicht verändert.

Wird zwar der OK-Knopf gewählt, aber keine Datei spezifiziert, so hat dies dieselbe Wirkung wie das Wählen des Abbruch-Knopfes.

Die Funktion "is_menu_key" wurde modifiziert. Das Zeichen für Shift ist nun nicht mehr "s", sondern der Pfeil nach oben. Menükürzel sollten also entsprechend angepaßt werden.

# Modul WINDOWS

In diesem Modul hat sich einiges getan. Dies belegt schon die Tatsache, daß knapp 40 KB Quellcode dazugekommen sind.

Zunächst wurden die Window-Flags erweitert. Es bedeuten:

WI_LOCKED: Das Fenster ist gesperrt. Dieses Flag wird intern benutzt, wenn modale Dialogboxen aufgerufen werden.

WI_FIRSTDRW: Das Fenster wurde zum erstenmal gezeichnet. Dieses Flag wird intern benutzt, wenn eine Dialogbox in einem Fenster, welches noch nicht gezeichnet wurde, bereits einen Mausklick erhält.

WI_DLCLOSE: Das Fenster, welches eine modale oder nicht-modale Dialogbox enthält, soll geschlossen werden. Dieses Flag wird intern benutzt, wenn der Anwender auf einen EXIT-Knopf in einer Dialogbox gedrückt hat. Dann wird das Fenster geschlossen. Soll dies verhindert werden, weil beispielsweise ein Fehlerzustand aufgetreten ist, so daß die Dialogbox noch weiter geöffnet bleiben soll, kann dieses Flag zurückgesetzt werden. Dies wird gewöhnlich in der Routine geschehen, welche auf einen Mausklick in der Dialogbox reagiert.

WI_MODAL: Das Fenster stellt eine modale Dialogbox dar. Eine modale Dialogbox muß zuerst verlassen werden, bevor das Programm seine weiteren Ausführungen fortsetzt. Mehr über modale Dialogboxen folgt in der Beschreibung zum Modul DIALOG.

WI_MODELESS: Das Fenster stellt eine nicht-modale (modeless) Dialogbox dar. Eine nicht-modale Dialogbox existiert wie ein Fenster parallel zum eigentlichen Programm. Mehr über modale Dialogboxen folgt in der Beschreibung zum Modul DIALOG.

WI_CURSKEYS: Das Fenster scrollt automatisch, wenn die Pfeiltasten gedrückt werden. Es gelten folgende Tastenbelegungen:

| | |
|---|---|
| UP: | Zeile nach oben |
| Shift-UP oder PGUP: | Seite nach oben |
| LEFT: | Zeile nach links |
| Shift-LEFT: | Seite nach links |
| RIGHT: | Zeile nach rechts |
| Shift-RIGHT: | Seite nach rechts |
| DOWN: | Zeile nach unten |
| Shift-DOWN oder PGDOWN: | Seite nach unten |
| POS1 (HOME): | Anfang des Dokumentes |
| END: | Ende des Dokumentes |

WI_MNSCROLL: Das Fenster hat eine scrollbare Menüzeile. Ist eine Menüzeile in ein Fenster eingeklinkt, so ist sie nicht scrollbar, wenn dieses Flag nicht angegeben wird. Bei kurzen Menüzeilen kann auf die Pfeile in den Menüs verzichtet werden.

WI_TOPMENU: Das Menü des Fensters soll nur funktionieren, wenn das Fenster ganz oben liegt. Im Normalfall kann auch das Menü eines unten liegenden Fensters mit der rechten Maustaste aktiviert werden. Bei Fenstern, die bestimmte Dinge nur tun können, wenn das Fenster oben liegt, kann es angebracht sein, mit Hilfe dieses Flags zu verhindern, daß das Menü bedient werden kann, wenn das Fenster nicht oben liegt.

Der Window-Typ wurde ebenfalls erweitert. Er bietet 5 neue Komponenten:

WORD subclass;

Die Unterklasse des Fensters. Es handelt sich um einen weiteren Bezeichner, der Fenster unterscheiden kann. Man geht davon aus, daß man zwei Fenster hat, die

fast das gleiche erledigen, wobei ein Fenster z.B. nur eingeschränkte Fähigkeiten hat. Dann können beide Fenster die gleiche Klasse bekommen, aber verschiedene Unterklassen besitzen. Beim Aufruf von "create_window" wird die Unterklasse gleich der Klasse gesetzt. Danach kann sie von der Applikation geändert werden. Im Code, der für beide Fensterarten gleich ist, können nun mit Hilfe der "subclass" die Fähigkeiten eines Fensters z.B. eingeschränkt werden.

Um alle Fenster einer Unterklasse zu bekommen, kann man bei "search_window" und bei "num_windows" zusätzlich das Flag SRCH_SUB angeben, also z.B.

search_window (CLASS_TRASH, SRCH_OPENED | SRCH_SUB, NIL);

Der Aufruf sucht alle Papierkorbfenster der Unterklasse CLASS_TRASH. Die Klasse des Papierkorbs kann z.B. CLASS_LIST sein, weil der Papierkorb als Liste dargestellt wird.

WORD bg_color;

Die Hintergrundfarbe eines Fensters kann gesetzt werden. Dann wird der Hintergrund des Fensters beim Öffnen noch vor der ersten REDRAW-Message in dieser Farbe gezeichnet. Voreingestellt ist WHITE. Zum Unterdrücken des Zeichnens muß eine Zahl kleiner 0 (z.B. –1) angegeben werden.

WORD edit_obj;

Beinhaltet das aktuelle Objekt, in welchem sich der Cursor befindet, wenn eine modale oder nicht-modale Dialogbox in einem Fenster nach oben kommt. Ist der Wert <= 0, so befindet sich kein editierbares Objekt in dieser Dialogbox. Vor jedem Öffnen einer Dialogbox sollte der Wert auf das erste editierbare Objekt gesetzt werden.

WORD edit_inx;

Aktueller Index im editierten Objekt, d.h. der Buchstabe, hinter dem der Cursor steht. Vor jedem Öffnen einer Dialogbox sollte der Wert initialisiert werden. Dabei bedeutet 0, daß der Cursor sich auf dem ersten Buchstaben befindet. Wird der Wert auf NIL (–1) gesetzt, so wird der Cursor hinter den letzten Buchstaben gesetzt (Normalfall in GEM).

WORD exit_obj;

Aktuelles Objekt, in welches in einer modalen oder nicht-modalen Dialogbox geklickt wurde. Meistens handelt es sich um einen Button. Ist es ein EXIT-Button, so wird das Flag WI_DLCLOSE gesetzt und die Dialogbox geschlossen. Ausnahmen hierzu siehe die Erläuterung zum Flag WI_DLCLOSE. Mit Hilfe von "edit_obj" und "exit_obj" kann bei Auslösen eines Tastatur- oder Klick-Ereignisses einer Dialogbox entsprechend reagiert werden (siehe Beispiel weiter unten).

BOOLEAN any_open (BOOLEAN incl_desk, BOOLEAN incl_closer,
BOOLEAN incl_modal);

Es kann getestet werden, ob irgendwelche Fenster offen sind. Dabei kann noch spezifiziert werden, ob der Desktop auch zu den offenen Fenstern zählt (er kann meist nicht geschlossen werden), oder ob das Vorhandensein der Schließbox

berücksichtigt werden soll. Außerdem kann angegeben werden, ob modale Dialogboxen ebenfalls zu den offenen Fenstern zählen sollen.

any_open : TRUE, wenn mindestens ein Fenster dieses Prozesses offen ist, das auf die Spezifikation zutrifft, sonst FALSE
incl_desk : TRUE, wenn der Desktop ebenfalls berücksichtigt werden soll, sonst FALSE
incl_closer : TRUE, wenn nur Fenster berücksichtigt werden sollen, die auch eine Schließbox haben, sonst FALSE
incl_modal : TRUE, wenn modale Dialogboxen ebenfalls als offene Fenster gezählt werden sollen, sonst FALSE

Von dieser Funktion kann z.B. das Ein/Ausschalten eines Menüs, welches "Schließen" heißt, abhängig gemacht werden (siehe auch Modul MENU).

WORD num_locked (VOID);

Zählt die Anzahl der gesperrten Fenster. Fenster werden gesperrt, wenn eine modale Dialogbox (auch Alertbox) geöffnet wird. Die einzigen nicht gesperrten Fenster sind dann die Dialogbox selbst und ein eventuell vorhandenes Hilfe-Fenster. Außerdem natürlich sämtliche Fenster von anderen Prozessen, die nach wie vor bedient werden können.

num_locked : Anzahl der gesperrten Fenster

Die Routine wird nur vom Modul DIALOG benutzt, um ganz sicher festzustellen, ob eine modale Dialogbox beendet wurde.

VOID cycle_window (VOID);

Holt das unterste Fenster nach oben, wenn es nicht der Desktop ist oder durch eine modale Dialogbox gesperrt ist.

VOID set_cursor (WINDOWP window, WORD obj, WORD inx);

Setzt den Cursor einer modalen oder nicht-modalen offenen Dialogbox auf ein bestimmtes editierbares Objekt.

window : Zeiger auf das Fenster, welches eine Dialogbox darstellt
obj: Nummer des editierbaren Objekts, auf das der Cursor gesetzt werden soll
inx: Index (Buchstabe) innerhalb des Objekts oder NIL, wenn der Cursor am Ende des Textes erscheinen soll

```
VOID drag_boxes (WORD num_objs, CONST RECT *boxes,
                 WINDOWP inv_window, SET inv_objs,
                 RECT *diff, CONST RECT *bound,
                 WORD x_raster, WORD y_raster);
```

Es können mehrere Rechtecke gleichzeitig verschoben werden. Außerdem kann ein Fenster angegeben werden, von welchem Objekte invertiert werden, die während des Zieh-Vorgangs überstrichen werden. Ein Raster (z.B. 8 für Byteraster) gibt an, ob die Rechtecke nur auf einem imaginären Raster erscheinen sollen.

num_objs : Anzahl der Objekte, die verschoben werden
boxes : Zeiger auf eine Rechteckliste, die "num_objs" Rechtecke enthält, welche die zu verschiebenden Rechtecke beschreiben
inv_window : Zeiger auf ein Fenster, von welchem Objekte während des Ziehens invertiert werden
inv_objs : Objekte, welche innerhalb dieses Fensters selektiert werden dürfen
diff : Zeiger auf ein Rechteck, welches die Differenz der verschobenen Rechtecke zum Ausgangspunkt angibt
bound : Zeiger auf ein Rechteck, in welchem sich die zu verschiebenden Rechtecke bewegen dürfen oder NULL, wenn sich diese innerhalb des gesamten Desktops bewegen dürfen
x_raster : X-Raster, auf dem sich die Rechtecke bewegen dürfen
y_raster : Y-Raster, auf dem sich die Rechtecke bewegen dürfen

Die Funktion simuliert im Prinzip "graf_dragbox" des AES. Allerdings dürfen hier mehrere Rechtecke angegeben werden. Außerdem können noch Objekte eines Fensters invertiert werden. Das Zeichnen der Rechtecke ist eine nichttriviale Aufgabe, da diese auf jedem Hintergrund korrekt aussehen sollen (schwarze Linien). Dies gilt vor allem für das Muster des Desktop. Je nach Position des Rechtecks und Zeichenalgoritmus der vier Linien müssen andere Linienmuster benutzt werden. Dieser Algorithmus funktioniert zumindest für alle Rechner, bei denen der linke obere Eckpunkt (0, 0) beim Desktop-Muster nicht gesetzt ist. Falls ein Grafikprozessor seine Arbeit verrichtet, der das "Grau" des Desktop invertiert darstellt, bzw. andere Algorithmen benutzt, um die Linien zu zeichnen, ist die Routine natürlich nicht mehr ganz korrekt. Ein Programm, das z.B. einen unkorrekten Algorithmus verwendet, ist das Resource-Construction-Set von Kuma. Bewegt man dort Piktogramme, so erscheinen je nach deren Position abwechselnd schwarze und weiße Umrisse.

Als Ergebnis wird in "diff.w" und "diff.h" die Differenz der Boxen nach dem Ziehvorgang zurückgegeben. Darüberhinaus ist in "diff.x" und "diff.y" die letzte Mausposition gespeichert. Beispiele für diese Funktion finden sich in den Modulen CLIPBRD und DESKTOP.

VOID draw_listobj (LISTBOX *list, WORD obj, BOOLEAN flip);

Zeichnet ein Element einer Listbox. Dabei kann noch angegeben werden, ob das Element umgeschaltet werden soll (von selektiert auf nicht selektiert und umgekehrt). Die Programmierung von Listboxen einschließlich der Listbox-Struktur wird weiter unten erläutert.

list : Zeiger auf eine Listbox-Struktur
obj : Objekt, welches gezeichnet werden soll
flip : TRUE, wenn das Element umgeschaltet werden soll

BOOLEAN listbox (LISTBOX *list, UWORD flags, MKINFO *mk);

Verwaltung einer Listbox. Je nach Angabe eines Flags können drei unterschiedliche Aufgaben angefordert werden: die Initialisierung, das Zeichnen und das Hineinklicken.

list : Zeiger auf eine Listbox-Struktur
flags : Einer der folgenden Werte:
LIST_INIT : Die Listbox wird initialisiert
LIST_DRAW : Die Listbox wird gezeichnet (nur mit LIST_INIT)
LIST_CLICK : Es wurde in die Listbox geklickt
mk : Zeiger auf eine Maus-Keyboard-Info-Struktur, falls "flags" LIST_CLICK ist, sonst NULL

Das Flag LIST_INIT wird benutzt, um die Listbox zu initialisieren. Dies kann auch mehrere Male geschehen (siehe Beispiel in Modul MENU).

Das Flag LIST_DRAW wird nur in Zusammenhang mit LIST_INIT benutzt. Damit kann man erreichen, daß diese Listbox beim Initialisieren sofort gezeichnet wird. Dies sollte man tun, wenn sich die Listbox bereits auf dem Bildschirm befindet und ihre Werte ändert (siehe Beispiel in Modul MENU). Wird das Flag mit LIST_INIT benutzt, so muß es mit diesem in einer ODER-Operation verbunden werden (LIST_INIT | LIST_DRAW). Das Zeichnen selbst geschieht ansonsten durch das Zeichnen des Fensters, in dem sich die Dialogbox befindet.

Wird das Flag LIST_CLICK angegeben, so muß in "mk" ein Zeiger auf die aktuellen Werte der Maus und des Keyboards übergeben werden. Diese erhält man als Parameter, wenn in ein Fenster geklickt wird (siehe Beispiel in Modul MENU).

VOID edit_noecho (MKINFO *mk, WORD cursor, BYTE *s, WORD maxlen);

Die Routine wird benötigt, um ein Paßwort in einer Dialogbox zu editieren.

mk : Zeiger auf eine MKINFO-Struktur, aus der der ASCII-Code und der Scan-Code einer Taste benutzt werden
cursor : Aktuelle Position des Schreibcursors
s : Zeiger auf die versteckte Zeichenkette
maxlen : Maximale Anzahl der Zeichen in der versteckten Zeichenkette

Die Routine muß immer aufgerufen werden, wenn in einer Dialogbox eine versteckte Zeichenkette wie z.B. ein Paßwort editiert werden kann. Der Grund dafür liegt darin, daß im editierbaren GEM-Objekt nur Sternchen '*' erscheinen sollen. Damit diese auch beim Wiederzeichnen richtig gehandhabt werden (Dialogboxen können überlagert werden), müssen im editierbaren Objekt tatsächlich nur '*'-Symbole vorhanden sein. Die eigentliche Information (z.B. das Paßwort) muß an einer anderen Stelle abgelegt werden (z.B. als statische Zeichenkette im Hauptspeicher).

Jedesmal, wenn eine Taste gedrückt wird, wird nun vom Window-Manager ein '*' in das GEM-Objekt eingetragen. Der Programmierer muß nun einen zusätzlichen Aufruf tätigen, um das eigentliche Paßwort im Speicher zu manipulieren. Dies nimmt die Routine "edit_noecho" ab. Sie reagiert auch auf Sonderzeichen, ESC, BS oder DEL, so daß in der versteckten Zeichenkette ganz normal editiert werden kann.

Muß das Paßwort nun ausgewertet werden, so darf nicht "get_ptext" benutzt werden, da dieser Aufruf nur die '*'-Symbole liefern würde. Stattdessen muß die statische Zeichenkette benutzt werden, in der zu jedem Zeitpunkt das aktuelle Paßwort steht.

Bei der Programmierung muß außerdem darauf geachtet werden, daß zu jedem Paßwort auch eine statische Variable angelegt wird, die den letzten Index des Cursors in diesem Paßwort angibt. Der Index wird zwar in der WINDOW-Struktur festgehalten (edit_inx), aber nach dem Drücken beispielsweise einer Pfeil-rechts Taste wird dieser Index um eins erhöht. Dann ist der Wert des alten Index nicht mehr aktuell. Dieser muß aber benutzt werden, um die letzte Aktion in der Routine "edit_noecho" nachzuführen. Im Modul MENU wird dafür ein Beispiel angegeben.

```
BOOLEAN init_windows (WORD err_nowindow, WORD maxreswind,
                      WORD class_help);
```

Das Modul WINDOWS wird initialisiert. Dabei wird eine Konstante für die Fehlermeldung "Keine weiteren Fenster mehr" und die Anzahl maximal zu erwartender residenter Fenster übergeben. Außerdem wird die Klasse der Hilfe-Fenster festgelegt.

| | |
|---|---|
| init_windows | : TRUE, wenn die Initialisierung geklappt hat, sonst FALSE |
| err_nowindow | : Index der Alertbox, die vom Window-Manager aufgerufen werden soll, wenn keine weiteren Fenster mehr erzeugt werden können oder NIL, wenn keine Meldung erscheinen soll |
| max_reswind | : Maximale Anzahl der zu erwartenden residenten Fenster |
| class_help | : Nummer der Klasse eines Fensters, welche alle Hilfetexte ausgibt (wird z.B. in der Datenbank PHOENIX benutzt). |

Die Initialisierung läuft schief, wenn nicht mehr genügend Speicher zur Verfügung steht, um genügend Strukturen des Typs WINDOW zur Verfügung zu stellen. Die Anzahl der maximal gleichzeitig zur Verfügung stehenden Strukturen werden in "max_reswind" übergeben. Eine Struktur bleibt resident im Speicher, wenn ein Fenster beim Schließen nicht gelöscht wird. Dies ist der Fall, wenn das Flag "WI_RESIDENT" gesetzt ist und es keine Möglichkeit gibt, das Fenster komplett zu löschen (etwa durch eine Piktogramm-Operation).

Die Anzahl errechnet sich folgendermaßen: Da in GEM maximal sieben Fenster der gleichen Klasse geöffnet werden können, können pro Klasse, bei der die Fenster resident bleiben, auch maximal sieben Strukturen im Speicher stehen. Von den Fenstern, die nicht resident bleiben, können maximal sieben gleichzeitig geöffnet werden. Da beim Schließen auch die Datenstruktur freigegeben wird, kann sie für das nächste Fenster genutzt werden. Faustregel ist also

7 * "Anzahl der residenten Klassen" + 7 + Anzahl Dialogboxen

Ausnahmen sind Klassen, bei denen nur maximal ein Fenster existieren kann. Dies sind z.B. der Desktop oder das Papierkorb- bzw. Klemmbrettfenster. In unserer Beispielapplikation können maximal 21 Fenster existieren (maximal sieben davon offen bzw. acht bei Berücksichtigung des Desktops):

1 Desktop-Fenster
1 Papierkorb-Fenster
1 Klemmbrett-Fenster
7 Grafik-Fenster

Das sind zehn Fenster, die resident im Speicher gehalten werden. Dazu kommen maximal sieben weitere Fenster, die nicht resident sind und deswegen nur in geöffnetem Zustand existieren:

7 Meta-Datei-Fenster oder
7 Bit-Image-Datei-Fenster oder
7 Text-Fenster oder
7 Fenster mit mathematischen Potenzen

Diese letzteren vier Klassen sind nicht resident, so daß eine beliebige Kombination von ihnen (zusammen maximal sieben) gleichzeitig offen sein kann.

Zu diesen Fenstern kommen alle Dialogboxen (vier bei unserem Programm), da diese ebenfalls resident gehalten werden. Dadurch wird erreicht, daß sich die Dialogboxen an der Stelle öffnen, an der sie zum letztenmal geschlossen wurden. Möchte man dies nicht, kann man das Flag WI_RESIDENT bei Dialogboxen ausblenden. In diesem Fall muß man auch nur ein Fenster für obige Formel berechnen.

Die Anzahl der Fenster hätte auch dynamisch ermittelt werden können, indem bei jedem Öffnen eines Fensters dynamisch für diese eine Fensterstruktur Speicherplatz reserviert werden würde. Das hätte aber eine Zerstückelung des Hauptspeichers zur Folge (besonders bei Accessories), was wir nicht anstreben.

Falls man bei der Berechnung Schwierigkeiten hat, sollte man lieber eine größere Zahl angeben. 50 Fenster wird wohl niemand so schnell erreichen. Die Größe einer Fensterstruktur ist sizeof (WINDOW), was z. Zt. 334 Bytes entspricht.

# Modul EVENT

Im Modul EVENT ist nun auch die Unterstützung des rechten Mausknopfes realisiert. Die folgenden globalen Variablen sind erreichbar.

GLOBAL UWORD events;

Gibt die aktuellen Ereignisse an, auf die gewartet werden soll. Die Variable wird in "init_event" auf den Wert

MU_KEYBD | MU_BUTTON | MU_M1 | MU_MESAG | MU_TIMER

gesetzt. Soll ein Ereignis (z.B. MU_TIMER) nicht berücksichtigt werden, so kann die Variable nach Aufruf von "init_event" entsprechend modifiziert werden. Das Ereignis MU_M1 sollte allerdings nur ausgeblendet werden, wenn kein Fenster eine eigene Mausform hat.

GLOBAL LONG millisecs;

Gibt die aktuellen Millisekunden an, auf die bei einem Zeitereignis gewartet werden soll. Die Variable wird mit 100 initialisiert.

GLOBAL WORD bclicks;

Gibt die Klicks an, auf die gewartet werden soll. Früher wurde hier der Wert 2 eingetragen (Doppelklick). Setzt man allerdings das niederwertigste Bit im höherwertigen Byte (0x0102), so wird auf die Negation des Mausknopfereignisses gewartet. Dies bedeutet, wir können beim Button-Status (s.u.) den Wert 0 angeben. Dieser bedeutet eigentlich, warte auf Loslassen von Mausknopf 1 und 2 gleichzeitig, da "bmask" auf 3 gesetzt wird. Die boolesche Negation bedeutet hier also, warte auf Drücken von Mausknopf 1 oder 2. Aus Loslassen wird Drücken, aus "und" wird "oder". Dadurch kann statt auf das Drücken beider Knöpfe zusammen auf das Drücken jedes der beiden Knöpfe gewartet werden.

GLOBAL WORD bmask;

Maske, die angibt, auf welche Knöpfe gewartet werden soll. Die Variable wird auf 0x0003 initialisiert, was bedeutet, daß beide Mausknöpfe benutzt werden sollen.

GLOBAL WORD bstate;

Maske, die angibt, ob auf Drücken oder Loslassen gewartet werden soll. Durch die oben beschriebene Negation wird der Wert auf 0x0000 initialisiert und nicht mehr geändert. Zunächst wird somit auf das Drücken der Mausknöpfe gewartet. Ist dies eingetreten, wird durch Wegblenden des niederwertigsten Bits im höherwertigen Byte von bclicks erreicht, daß nun auf das Loslassen gewartet wird.

Die folgenden Routinen sind nun global verfügbar, da diese zum Teil im Modul DIALOG aufgerufen werden.

```
VOID hndl_events (VOID);
VOID hndl_keybd (MKINFO *mk);
VOID hndl_button (MKINFO *mk);
VOID hndl_mesag (WORD *msgbuff);
VOID hndl_timer (LONG millisecs);
```

Ihre Beschreibung kann weiter vorn nachgelesen werden.

# Modul DIALOG

Dieses neue Modul wurde konzipiert, um zusammen mit dem Modul WINDOWS modale und nicht-modale Dialogboxen sowie Alertboxen zu implementieren, die sich in Fenstern befinden. Dadurch wird eine wesentlich höhere Flexibilität erreicht. Beispielsweise kann bei jedem Tastendruck und Mausklick die Applikation entscheiden, wie in der Dialogbox reagiert werden soll. Auf diese Weise kann beispielsweise verhindert werden, daß eine Dialogbox mit einer Fehleingabe verlassen wird.

Die Programmierung solcher Alertboxen wird weiter unten erklärt.

Die folgenden Definitionen sind in der Headerdatei zu finden.

```
#define CLASS_DIALOG  3
```

Die Klasse eines Dialogfensters (Dialogbox). Dabei ist es unwesentlich, ob es sich um eine modale oder nicht-modale Dialogbox handelt. Auch Alertboxen zählen zu dieser Klasse.

```
#define NUM_SEP      5
#define SEP_OPEN     '['
#define SEP_CLOSE    ']'
#define SEP_LINE     'l'
```

Die vier Konstanten werden benutzt, um Fehlermeldungen zusammenzubauen. Das Format der Fehlermeldungen wird bei der Fehlerbehandlung beschrieben.

```
typedef BOOLEAN (*HELPFUNC) (BYTE *helpmsg);
```

Es handelt sich um den Typ einer Hilfe-Funktion die aufgerufen wird, wenn in einer Alertbox die Help-Taste gedrückt wird. Diese Hilfe-Funktion kann mittels der Funktion "set_helpfunc" (s. u.) registriert werden.

```
BOOLEAN init_dialog (BYTE **alerts, OBJECT *tree,
                     WORD index, BYTE *title);
```

Initialisiert das Modul DIALOG.

| | |
|---|---|
| init_dialog | : TRUE, wenn das Initialisieren geklappt hat, sonst FALSE |
| alerts | : Zeiger auf die Tabelle der Fehlermeldungen |
| tree | : Zeiger auf den Objektbaum, der die Alertbox repräsentiert |
| index | : Resource-Index des obigen Objektbaumes |
| title | : Zeiger auf die Titelzeile aller Fehlermeldungen |

Der Parameter "alerts" ist normalerweise die globale Variable aus GLOBAL.H.

Die Alertbox für den Parameter "tree" ist in der Resource-Datei SCRAP.RSC abgelegt. Für eigene Anwendungen kann er aus dieser kopiert werden.

Der Parameter "index" ist in der Datei SCRAP.H definiert. Es handelt sich um den Bezeichner für den Objektbaum.

In jeder Alertbox, die ja in einem echten Fenster dargestellt wird, gibt es eine Titelzeile (Parameter "title"). Sie soll den Namen der Applikation wiedergeben. Sie wird nur einmal festgelegt und ändert sich im Laufe des Programmablaufs nicht mehr. Damit wird gewährleitet, daß Fehlermeldungen von verschiedenen Applikationen und Accessories identifiziert werden können. Man darf in diesem Zusammenhang nicht vergessen, daß beim Auftreten eines Fehlers zwar der aktuelle Prozeß "gestoppt" wird, bis die Fehlermeldung quittiert wird, jedoch können andere Appikationen oder Accessories bedient werden, wenn eines ihrer Fenster nach oben geholt wird.

Ein Aufruf von "init_dialog" könnte z.B. folgendermaßen aussehen:

```
ok = init_dialog (alerts, alert, ALERT, " SCRAP ");
```

```
BOOLEAN term_dialog (VOID);
```

Das Modul DIALOG wird terminiert.

term_dialog : TRUE, wenn das Terminieren geklappt hat, sonst FALSE.

VOID set_helpfunc (HELPFUNC help);

Setzt die Hilfe-Funktion für Alertboxen.

help : Zeiger auf eine Funktion, die aufgerufen wird, wenn in einer Alertbox der Hilfe-Knopf gedückt wird oder NULL, wenn es keine Hilfe-Funktion für Alertboxen gibt.

Die Hilfe-Funktion wird aufgerufen, wenn in einer Alertbox die Hilfe-Taste gedrückt wird und gleichzeitig ein Hilfe-Button definiert ist. Der Funktion wird dann eine Zeichenkette übergeben, durch die der genaue Kontext ersichtlich ist. Diese Zeichenkette kann vom Entwickler selbst bestimmt werden und befindet sich in der Datei SCRAP.ERR. Deren Aufbau wird bei der Fehlerbehandlung weiter unten beschrieben.

Der Aufruf von "set_helpfunc" könnte also folgendermaßen aussehen:

```
set_helpfunc (appl_help);
```

Die Funktion "appl_help" muß vorher deklariert weden und kann dann z.B. folgendermaßen aussehen:

```
LOCAL BOOLEAN appl_help (helpmsg)
BYTE *helpmsg;

{
  return (open_help ("SCRAP.HLP", helpmsg));
} /* appl_help */
```

Die Funktion "open_help" wiederum sollte sich in einem Modul HELP befinden. Ein solches Fenster hat die Klasse CLASS_HELP und wird damit beim Öffnen nicht gesperrt. Somit kann beliebig zwischen dem Alert-Fenster und dem Hilfe-Fenster umgeschaltet werden. Das Hilfe-Fenster könnte aus einer Hilfe-Datenbank, die z.B. den Namen SCRAP.HLP trägt, die übergebene Zeichenkette heraussuchen und den entsprechenden Hilfetext darstellen. Auf diese Weise wird bei der relationalen Datenbank PHOENIX verfahren. Die Hilfe-Datenbank liegt im PHOENIX-Format vor, so daß der Benutzer diese sogar editieren und erweitern kann.

WINDOWP crt_dialog (OBJECT *obj, OBJECT *menu, WORD icon,
BYTE *title, UWORD flags);

Kreiert eine modale oder nicht-modale Dialogbox in einem Fenster.

crt_dialog : Zeiger auf das kreierte Fenster oder NULL, wenn es nicht kreiert werden konnte
obj : Zeiger auf einen Objektbaum, der die Dialogbox repräsentiert
menu : Zeiger auf einen Objektbaum, der eine optionale Menüzeile repräsentiert oder NULL, wenn keine Menüzeile vorhanden sein soll
icon : Index des Objektbaumes "obj", welcher in der Resource-Header-Datei abgelegt ist. Dieser Wert wird in "window-icon" abgelegt

title : Fenster-Titel der Dialogbox
flags : Zusätzliche Flags, die das Verhalten des Fensters steuern. Standardmäßig ist WI_RESIDENT vorgegeben. Dazu kann kommen:

WI_MODAL : Eine modale Dialogbox soll erzeugt werden
WI_MODELESS : Eine nicht-modale Dialogbox soll erzeugt werden

Beim Kreieren muß man beachten, daß für modale Dialogboxen zur leichteren Identifizierung der Schließ-Knopf ausgeblendet wird. Zusätzlich sollte der Titel drei Punkte aufweisen.

Existiert ein Hilfe-Knopf (ist das HELP_FLAG gesetzt), so wird dieser Hilfe-Knopf abgeschaltet, wenn keine Hilfe-Funktion existiert.

Da das Fenster resident kreiert wird, wird eine Dialogbox immer an der Stelle erscheinen, an der sie zuletzt geschlossen wurde. Falls dies nicht gewünscht wird, kann nach dem Kreieren das Flag durch den Befehl

window-flags &= ~ WI_RESIDENT;

ausgeblendet werden.

BOOLEAN open_dialog (WORD icon);

Ein Dialogfenster (Dialogbox) wird geöffnet.

open_dialog : TRUE, wenn das Fenster geöffnet werden konnte, sonst FALSE
icon : Index des Objektbaumes der Dialogbox, welcher in der Resource-Header-Datei abgelegt ist. Dieser Wert wurde bei crt_dialog in "window-icon" abgelegt.

Kann die Dialogbox nicht geöffnet werden, so wird keine Fehlermeldung ausgegeben, da dieses Modul applikationsunabhängig programmiert ist. Die Fehlermeldung muß vom Aufrufer ausgegeben werden.

WORD hndl_alert (WORD alert_id);

Behandelt eine Fehlermeldung.

hndl_alert : Wert des Knopfes, der zur Beendigung des Fehlerdialoges geführt hat, wobei gilt: 1 = erster Knopf, 2 = zweiter Knopf usw.
alert_id : Nummer der Fehlermeldung aus ERRORS.H

Die Funktion ruft lediglich "open_alert" mit der entsprechenden Zeichenkette aus SCRAP.ERR auf.

WORD open_alert (BYTE *alertmsg);

Öffnet eine Alertbox und behandelt deren Darstellung und Interaktion.

open_alert : Wert des Knopfes, der zur Beendigung des Fehlerdialoges geführt hat, wobei gilt: 1 = erster Knopf, 2 = zweiter Knopf usw.
alertmsg : Zeiger auf eine Zeichenkette (z.B. aus der Datei SCRAP.ERR), welche das folgende Format aufweisen muß:

[Piktogramm][Fehlertext][Knöpfe][Belegung][Hilfetext]

Piktogramm kann die Werte 0 bis 3 annnehmen, wobei gilt:

0: Info-Piktogramm
1: Ausrufezeichen-Piktogramm
2: Fragezeichen-Piktogramm
3: STOP-Piktogramm

Der Fehlertext kann in einzelne Zeilen aufgespalten werden. Dabei werden die Zeilen mit dem Zeilentrenner (SEP_LINE = '|') getrennt. Es können maximal 5 Zeilen (NUM_SEP) mit jeweils maximal 40 Zeichen definiert werden. Ist eine Zeile zu lang, so wird sie abgeschnitten. Auf keinen Fall ragt sie fälschlicherweise in die nächste Zeile, wie dies bei den GEM-Alertboxen der Fall ist.

Der Text der Knöpfe wird wie der Fehlertext beschrieben. Maximal können 5 Knöpfe zu jeweils maximal 12 Buchstaben Platz finden. Nicht benutzte Knöpfe werden versteckt. Die restlichen Knöpfe werden zentriert dargestellt.

Die Default-Belegung der Knöpfe wird im nächsten Feld angegeben. Sie besteht immer aus vier Zahlen, die durch einen Separator ('|') getrennt werden. Die vier Zahlen bedeuten:

| | |
|---|---|
| Position 1: | 1 = Klingelzeichen mit der Fehlermeldung<br>2 = Kein Klingelzeichen mit der Fehlermeldung |
| Position 2: | Nummer des Default-Knopfes oder 0, wenn nicht vorhanden |
| Position 3: | Nummer des Abbruch-Knopfes oder 0, wenn nicht vorhanden |
| Position 4: | Nummer des Hilfe-Knopfes oder 0, wenn nicht vorhanden |

Im folgenden wird ein Beispiel für eine Alertbox gegeben. In einer Datei wie SCRAP.ERR stehen alle Alertmeldungen auf jeweils einer ganzen Zeile. Im folgenden Beispiel muß wegen der Länge die Zeile umbrochen werden.

```
[2]
[Drucker wird von anderem Prozess|
belegt. Soll der Prozess gespoolt|
oder auf die Freigabe des Druckers|
gewartet werden?]
[Spoolen|Warten|Abbruch|Hilfe]
[0|1|3|4]
[Warnung: Drucker belegt]
```

Wir haben also eine Alertbox mit Piktogramm 2 (Fragezeichen). Danach folgt der Text der Alertbox (vier Zeilen). Es gibt vier Knöpfe. Es soll kein Klingelzeichen ertönen (0), der Default-Knopf ist 1 (Spoolen), der Abbruch-Knopf ist 3 (Abbruch) und der Hilfe-Knopf ist 4 (Hilfe). Wird dieser betätigt, so wird die Hilfe-Funktion aufgerufen, wobei an sie die Zeichenkette "Warnung: Drucker belegt" übergeben wird. Die Hilfe-Funktion muß vorher über "set_helpfunc" registriert worden sein. Ist dies nicht der Fall, so wird der Hilfe-Knopf grau dargestellt.

Erscheint eine solche Alertbox auf dem Bildschirm, so kann sie nach Belieben verschoben werden, da es sich um ein Fenster handelt. Außerdem kann eine Hilfe-Funktion aufgerufen werden, welche den eigentlichen Programmfluß nicht stört. Möchte man die Alertbox nur mit der Tastatur bedienen, so kann man die Return-Taste, die Undo-Taste und die Help-Taste verwenden. Außerdem besteht

die Möglichkeit, durch die Tab-Taste bzw. durch Shift-Tab den Default-Knopf zu verändern. Dieser wandert dann immer um eine Position weiter.

Alertboxen sind nicht resident, so daß eine neue Alertbox immer in der Mitte des Schirmes erscheint.

Tritt der Extremfall auf, daß keine Fenster mehr zur Verfügung stehen (7 Fenster bereits geöffnet), kann eine solche Alertbox nicht in einem Fenster dargestellt werden. Da nun auch die Fehlermeldung "Das Objekt kann nicht geöffnet werden!" nicht erscheinen darf (Rekursion), wird die Alertbox als normale GEM-Dialogbox dargestellt. In diesem Fall wird der Hilfe-Knopf abgeschaltet und die Tabulator-Steuerung sowie die Undo-Taste funktionieren nicht mehr.

GLOBAL BOOLEAN set_alert (BOOLEAN as_dialog);

Setzt den Alertmodus. Damit kann bestimmt werden, ob Alertboxen im Fenster oder als GEM-Dialogbox erscheinen sollen.

| | |
|---|---|
| **set_alert** | : Liefert den alten Alertmodus, also TRUE, wenn Alertboxen als Dialogboxen behandelt werden sollen, sonst FALSE |
| **as_dialog** | : TRUE, wenn Alertboxen als GEM-Dialogboxen behandelt werden sollen, sonst FALSE |

Das Setzen des Alertmodus ist nur nötig, wenn beim Neuzeichnen eines Fensters ein Fehler auftreten kann. In diesem Fall würde durch eine normale Fehlerbehandlung erreicht werden, daß beim Schließen der Alertbox eine erneute REDRAW-Message des "fehlerhaften" Fensters gesendet würde. Diese würde durch das Neuzeichnen eventuell wieder eine Alertbox anstoßen usw. Durch das Setzen des Alertmodus wird erreicht, daß keine neue REDRAW-Message gesendet wird, da der Bildschirmhintergrund der nicht-bewegbaren Dialogbox gerettet wird.

Ein Setzen des Alertmodus kann ebenfalls sinnvoll sein, wenn man Statusmeldungen in einer gewöhnlichen Dialogbox ausgibt und während dieser Meldung ein Fehler auftreten kann. Da diese gewöhnliche Dialogbox keine REDRAW-Message erhält, würde der Bildschirmhintergrund nicht richtig restauriert werden.

Beim Setzen des Alertmodus geht man so vor, daß man sich in einer Variablen den alten Wert merkt, und diesen wieder hineinschreibt.

Beispiel:

```
{
  BOOLEAN old_mode;

  old_mode = set_alert (TRUE);

  /* kritischer Bereich...  */

  set_alert (old_mode);
}
```

VOID hndl_modal (BOOLEAN use_timer);

Behandelt modale Dialogboxen und Alertboxen. Dabei kann angegeben werden, ob der Timer ebenfalls berücksichtigt werden soll.

use_timer : TRUE, wenn der GEM-Timer berücksichtigt werden soll, sonst FALSE

Die Routine hat eine eigene "evnt_multi" Schleife, die terminiert, wenn die Situation, die zum modalen Zustand geführt hat, beendet ist. Eine solche Situation ist z.B. das Öffnen einer modalen Dialogbox oder einer Alertbox. Wird diese geschlossen, so terminiert die Schleife und das Programm wird an der Aufrufstelle fortgesetzt. In der Zwischenzeit können aber beliebige Nachrichten eintreffen, wie z.B. das Neuzeichnen eines Fensters. Diese werden ebenfalls berücksichtigt.

Nicht berücksichtigt werden Menü-Ereignisse (die Menüzeile ist beim modalen Zustand abgeschaltet) und eventuell Timer-Ereignisse, da während dieser Ereignisse Fehlermeldungen auftreten können. Insbesondere kann dieses bei Abarbeitung eines Fehlers sein. Es darf aber nur eine Alertbox auf dem Bildschirm sein, da nur eine Resource dafür existiert. Aus diesem Grund wird eine Alertboxbehandlung ohne Timer durchgeführt.

Ruft man selbst "hndl_modal" auf, nachdem man ein Fenster geöffnet hat, so ist dieses Fenster modal (es sperrt alle anderen Fenster bis auf das Hilfe-Fenster) und erst nach Schließen desselben wird im Programm weiterverfahren. In diesem Fall kann man "hndl_modal" mit dem Parameter TRUE aufrufen. In der Praxis wird man aber auf einen Aufruf von "hndl_modal" ganz verzichten können, da dies intern schon für modale Dialogboxen und Alertboxen erledigt wird.

Ist übrigens eine modale Dialogbox geöffnet, so dürfen daraus weitere modale Dialogboxen oder Alertboxen geöffnet werden. Der modale Zustand wird durch einen jeweils neuen Aufruf von "hndl_modal" gestapelt, so daß erst nach dem Schließen der letzten modalen Dialogbox das Programm an der Ursprungsstelle weiterläuft. Bei jedem Aufruf von "hndl_modal" wird die aktuelle Mausform gerettet und wieder restauriert.

# Modul INITERM

In diesem Modul wurden kleinere Änderungen vorgenommen, die aber wichtig sind.

Die Konstante CLASS_HELP wird als die Ziffer 2 definiert. Normalerweise wird die Klassenkonstante in der Headerdatei des entsprechenden Moduls definiert (z.B. HELP.H). Hier haben wir aber aus Platzgründen auf eine Implementierung eines eigenen Hilfe-Moduls verzichtet.

Die Konstante ALERT_NAME wird als Zeichenkette "SCRAP.ERR" definiert. Sie gibt an, wo sich die Datei mit den Fehlermeldungen befinden soll. Sie wird dann mittels "read_alerts", einer lokalen Funktion, eingelesen.

Der Aufruf "init_windows" wurde geändert:

```
ok &= init_windows (NOWINDOW, MAX_RESWIND, CLASS_HELP);
```

Als letzter Parameter soll hier die Klasse des Hilfe-Fensters angegeben werden. Ein echtes Hilfe-Fenster wurde hier aber nicht implementiert. Es handelt sich um ein gewöhnliches Fenster, welches Hilfetexte ausgeben kann. Im Modul WIN-

DOWS wird ein Fenster dieser Klasse aber nicht gesperrt, wenn z.B. eine modale Dialogbox oder eine Alertbox offen ist, so daß man eine kontextsensitive Hilfe zu jedem Zeitpunkt implementieren kann. Ein Beispiel für eine solche Hilfe zeigt das relationale Datenbanksystem PHOENIX. Dort kann zu jedem Zeitpunkt die Help-Taste bzw. F1 gedrückt werden und es erscheint ein Hilfe-Fenster, das beliebig manipuliert werden kann – auch wenn eine modale Dialogbox oder eine Alertbox offen ist.

Der Aufruf "init_dialog" wurde hinzugefügt:

```
ok &= init_dialog (alerts, alert, ALERT,
                   (BYTE *)freetext [FDESKNAM].ob_spec);
```

Seine Parameter wurden bereits im Modul DIALOG erklärt.

# ERRORS.H

Die Datei ERRORS.H nimmt die Konstanten aller Fehlermeldungen auf, die in dem Programm vorkommen können. Die Fehlermeldungen müssen bei 0 beginnen und fortlaufend numeriert sein. Wer möchte, kann dies auch mit einem enum-Feld erreichen, so sein Compiler dessen mächtig ist.

# Modul RESOURCE

Die benutzerdefinierten Objekte (extended object types, ob_type = G_USERDEF) haben sich geändert. Folgende 3 Typen stehen zur Verfügung:

```
#define DHEADER     0x0020
#define DHEIGHT     0x0040
#define DCRBUTTON   0x0080
```

DHEADER wird benutzt, um Überschriften für Rahmen zu erzeugen. Der Text wird um die halbe Höhe des Systemzeichensatzes nach oben verschoben, was in jeder Auflösung funktioniert.

DHEIGHT wird benutzt, um ein Objekt um die halbe Höhe des Systemzeichensatzes kleiner zu machen. Die Knöpfe der Dialogboxen werden mit diesem Typ versehen, damit sie nicht zu groß erscheinen. Sie sind dann 1.5 Zeichen hoch. Auch die Dialogboxen selbst (ROOT) werden mit diesem Typ versehen, wenn am unteren Rand Knöpfe erscheinen. Da die Knöpfe ja etwas kleiner sind, sollte die Dialogbox selbst ebenfalls kleiner sein, um den Abstand zum unteren Rand nicht zu groß werden zu lassen.

DCRBUTTON wird benutzt, wenn Objekte Checkboxes oder Radiobuttons sein sollen. Der Wert ob_flags eines Objekts muß den Wert RBUTTON beinhalten (wie das bei "normalen" Radiobuttons ja auch gemacht wird), wenn ein runder Radiobutton gezeichnet werden soll. Ist RBUTTON nicht gesetzt, so handelt es sich um eine Checkbox. Checkboxen werden im Dialogfenster "Einstellungen" benutzt.

Beispiele:

- Ein Knopf soll um eine halbe Zeichensatzhöhe kleiner gemacht und um denselben Betrag nach oben verschoben werden. Das obere Byte von ob_type muß den Wert DHEADER + DHEIGHT = 0x60 = 96 haben.
- Ein Radiobutton muß in ob_flags den Wert RBUTTON besitzen und im oberen Byte von ob_type den Wert DCRBUTTON = 0x80 = 128.
- Soll der Radiobutton von b) noch um eine halbe Zeile nach oben verschoben werden, so ist in ob_type der Wert DHEADER + DCRBUTTON = 0xA0 = 160 zu setzen.

Das obere Byte in ob_type kann in allen neuen Resource Construction Sets leicht gesetzt werden. Es heißt dort "Extended type" oder "Extended Object Type".

Neu ist auch das Setzen von Abbruch/Hilfe-Attributen für die Knöpfe von Dialogboxen. Außerdem kann das Attribut NO_ECHO gesetzt werden. Im Beispiel werden die Knöpfe der Settings-Dialogbox initialisiert.

```
do_flags (settings, SPASSWD, NOECHO_FLAG);
do_flags (settings, SCANCEL, UNDO_FLAG);
do_flags (settings, SHELP,   HELP_FLAG);
```

So kann das System bei Drücken der UNDO- bzw. HELP-Taste den richtigen Knopf aus einer Dialogbox auswählen. Zusätzlich kann der Window-Manager das Echo des Passwortes unterdrücken.

# Programmierung von Dialogboxen in Fenstern

Die Möglichkeiten der Dialoggestaltung wurden erheblich erweitert. Es ist jetzt möglich, Dialogboxen in Fenstern darzustellen. Dadurch können Dialoge frei auf dem Bildschirm bewegt werden.

Des weiteren können sogenannte nicht-modale Dialoge erstellt werden. Das sind Dialogfenster, die es erlauben weiterzuarbeiten, auch wenn der Dialog noch nicht beendet ist. Durch das einfache Setzen eines Flags (WI_MODELESS bzw. WI_MODAL) kann der Modus bestimmt werden.

Die beiden Dialogfenster "Einstellungen" sowie "Schriftauswahl" im Modul MENU sind nicht-modale Dialoge, die "About"-Box ist ein modaler Dialog.

Dadurch, daß man in beiden Dialogfensterarten (modal und modeless) bei jedem Tastendruck und bei jedem Mausklick vom Modul WINDOWS aufgerufen wird, besteht die Möglichkeit, Dialogfenster "kontextsensitiv" zu beeinflussen. Vorbei sind die Zeiten, in denen man das Flag "TOUCHEXIT" setzen mußte, um den Dialog kurzzeitig zu verlassen, wenn man z.B. den OK-Button "grau" machen wollte, weil eine Bedingung nicht mehr erfüllt ist. Das geht jetzt alles viel eleganter.

Da Dialogfenster wie "normale" Fenster behandelt werden, kann man in die Fensterstruktur (struct WINDOW) alle zur Verfügung stehenden Funktionen zur

Laufzeit einhängen, wenn diese benötigt werden. Meistens werden es die folgenden Funktionen sein:

- wi_click
- wi_key
- wi_showhelp
- wi_open
- wi_close
- wi_draw (seltener)

Als Beispiel dient hier das Dialogfenster für die Einstellungen. Zunächst folgt das Listing und dann die Beschreibung der einzelnen Funktionen.

```
LOCAL VOID get_settings ()
{
  STRING s;
  get_ptext (settings, SBLINK, s);
  sscanf (s, "%d", &blinkrate);

  ring_bell   = get_checkbox (settings, SBEEP);
  grow_shrink = get_checkbox (settings, SGROW);
} /* get_settings */

/*****************************************************************/

LOCAL VOID set_settings ()

{
  STRING s;

  sprintf (s, "%d", blinkrate);
  set_ptext (settings, SBLINK, s);

  set_checkbox (settings, SBEEP, ring_bell);
  set_checkbox (settings, SGROW, grow_shrink);

  undo_state (settings, SOK, DISABLED);
} /* set_settings */

/*****************************************************************/

LOCAL VOID click_settings (window, mk)
WINDOWP window;
MKINFO  *mk;

{
  switch (window->exit_obj)
  {
    case SOK     : get_settings ();
                   break;
    case SCANCEL : set_settings ();
                   break;
    case SHELP   : help_settings (NULL, NIL);
                   undo_state (window->object, window->exit_obj,
                               SELECTED);
                   draw_object (window, window->exit_obj);
                   break;
  } /* switch */
} /* click_settings */
```

```
/*****************************************************************/

LOCAL BOOLEAN key_settings (window, mk)
WINDOWP window;
MKINFO  *mk;

{
  BYTE *p;

  switch (window->edit_obj)
  {
    case SBLINK :
         p = ((TEDINFO *) settings [SBLINK].ob_spec)->te_ptext;
         if ((*p == EOS) == ! is_state (settings, SOK, DISABLED))
         {
           flip_state (settings, SOK, DISABLED);
           draw_object (window, SOK);
         } /* if */
         break;
  } /* switch */

  return (FALSE);
} /* key_settings */

/*****************************************************************/

LOCAL BOOLEAN help_settings (window, icon)
WINDOWP window;
WORD    icon;

{
  BOOLEAN ok;
  WINDOWP helpwin;

  helpwin = search_window (CLASS_DIALOG, SRCH_ANY, SETHELP);

  if (helpwin == NULL)
  {
    sethelp->ob_x = desk.x + desk.w - sethelp->ob_width;
    sethelp->ob_y = desk.y + desk.h - sethelp->ob_height;

    helpwin = crt_dialog (sethelp, NULL, SETHELP,
              (BYTE *)freetext [FHELPSET].ob_spec, WI_MODELESS);
  } /* if */

  ok = helpwin != NULL;

  if (ok)
  {
    if (helpwin->opened == 0)
    {
      helpwin->work.x = helpwin->scroll.x =
                        desk.x + desk.w - sethelp->ob_width;
      helpwin->work.y = helpwin->scroll.y =
                        desk.y + desk.h - sethelp->ob_height;
    } /* if */

    if (! open_dialog (SETHELP)) hndl_alert (ERR_NOOPEN);
  } /* if */

  return (ok);
} /* help_settings */
```

```
/*****************************************************************/

LOCAL VOID msettings ()

{
  WINDOWP window;
  WORD    ret;

  window = search_window (CLASS_DIALOG, SRCH_ANY, SETTINGS);

  if (window == NULL)
  {
    form_center (settings, &ret, &ret, &ret, &ret);
    window = crt_dialog (settings, NULL, SETTINGS,
                         (BYTE *)freetext (FSETTING].ob_spec,
                         WI_MODELESS);

    if (window != NULL)
    {
      window->click    = click_settings;
      window->key      = key_settings;
      window->showhelp = help_settings;

      undo_state (window->object, SHELP, DISABLED);
    } /* if */
  } /* if */

  if (window != NULL)
  {
    if (window->opened == 0)
    {
      window->edit_obj = find_flags (settings, ROOT, EDITABLE);
      window->edit_inx = NIL;

      set_settings ();
    } /* if */

    if (! open_dialog (SETTINGS)) hndl_alert (ERR_NOOPEN);
  } /* if */
} /* msettings */

/*****************************************************************/
```

LOCAL VOID msettings ()

Zunächst wird nachgesehen (search_window), ob das Dialogfenster schon existiert. Dies ist der Fall, wenn es zuvor schon geöffnet wurde. Existiert noch keines, so wird es zentriert und dann ein neues Fenster erzeugt.

Der erste Parameter ist immer der Objektbaum, der ja im RCS erstellt wurde.

Der zweite Parameter enthält den Objektbaum einer Menüleiste. Dialogfenster können also auch mit Menüleisten ausgestattet werden (die Funktionen updt_menu und hndl_menu müssen dann eingehängt werden, wie dies beim Modul CLIPBRD gemacht wurde).

Der dritte Parameter ist die Konstante, die im RCS für den Objektbaum angegeben wurde (SETTINGS).

Der vierte Parameter ist die Überschrift für das Dialogfenster. Sie erscheint in der Titelzeile des Fensters.

Der letzte Parameter ist das Flag, das angibt, ob der Dialog "modal" oder "nicht-modal" (modeless) ausgeführt werden soll. In obigem Beispiel WI_MODELESS.

Konnte das Dialogfenster erzeugt werden, so werden die Funktionen click, key und help eingehängt. Dadurch werden bei einem Mausklick in das Dialogfenster die Funktion click_settings, bei einer Taste die Funktion key_settings und bei Drücken der Hilfe- bzw. F1-Taste die Funktion help_settings aufgerufen.

Falls es wie hier eine Hilfe gibt, wird der Hilfe-Knopf selektierbar. (undo_state...DISABLED).

Ist das Fenster gerade geschlossen (opened == 0), so werden noch 3 Initialisierungen vorgenommen. Das edit_obj wird so eingestellt, daß es auf das erste Objekt zeigt, das editierbar (EDITABLE) ist. Der edit_inx wir auf NIL gesetzt, was bedeutet, daß der Cursor für das erste Edit-Objekt – wie in Dialogboxen üblich – ans Ende plaziert wird. Hier könnte man den Cursor auch anders einstellen.

Schließlich wird die Funktion set_settings aufgerufen, die die Werte aus den globalen Variablen (blinkrate, ring_bell und grow_shrink) in die GEM-Objekte überträgt. Die inverse Funktion dazu lautet get_settings. Sie holt die Werte aus der Dialogbox und speichert sie in den entsprechenden globalen Variablen ab.

Diese beiden Funktionen bekommen noch eine wichtige Bedeutung in der Funktion key_settings (besonders bei nicht-modalen Dialogfenstern).

Sind die Initialisierungen vorgenommen, wird das Dialogfenster geöffnet (open_dialog). Bei einem Fehler wird eine entsprechende Fehlermeldung über hndl_alert aufgerufen.

Achtung! Ist der Dialog nicht-modal, so kehrt die Funktion open_dialog sofort zurück, womit auch die Funktion msettings beendet ist. Bei "modalen" Dialogen wird von open_dialog erst zurückgekehrt, wenn der Dialog beendet ist (z.B. der OK-Knopf wurde gedrückt). In diesem Fall könnte man noch einige Aktionen unternehmen, je nachdem wie der Dialog beendet wurde. Z.B. könnte msettings einen Funktionswert zurückliefern.

LOCAL VOID get_settings (VOID)

Wie oben erwähnt holt diese Funktion alle Werte aus einem Objektbaum und legt sie an geeigneter Stelle ab.

LOCAL VOID set_settings (VOID)

Diese Funktion legt die benötigten Werte im Objektbaum ab. Dafür eignen sich besonders die Funktionen set_ptext, set_checkbox (für Checkboxen) sowie set_rbutton (für Radiobuttons). Hier wird auch der OK-Knopf zurückgesetzt, falls dieser vorher "grau" (DISABLED) war. Das kann dann der Fall sein, wenn der Dialog vorher in einem ungültigen Zustand mit dem ABBRUCH-Knopf verlassen wurde. Der OK-Knopf war in diesem Fall grau. Dies tritt ein, wenn der Wert in der Blinkrate gelöscht wird.

LOCAL VOID click_settings (WINDOWP window, MKINFO *mk)

Diese Funktion wird vom Modul WINDOWS aufgerufen, wenn in die Dialogbox geklickt wurde. Das Dialogfenster (window), sowie die Maus/Keyboard-Informationen (mk) werden übergeben. Das exit-Objekt des Fensters ist das Objekt, auf das man geklickt hat. In einem switch-Statement können die Fälle behandelt werden, die wichtig erscheinen.

Wurde der OK-Knopf (SOK) gedrückt, so holt man die Werte aus der Dialogbox (get_settings). Ist der Dialog "modal" ist er danach beendet. Ist er nicht-modal und wurde mit der rechten Maustaste OK gedrückt, werden die Werte geholt und für das Gesamt-Programm über die globalen Variablen zur Verfügung gestellt, aber das Fenster bleibt geöffnet.

Entsprechendes gilt für den ABBRUCH-Knopf (SCANCEL). Die Funktion set_settings sorgt bei nicht-modalen Dialogen, bei denen mit der rechten Maustaste auf Abbruch gedrückt wurde dafür, daß die alten Werte wieder erscheinen. Teste Sie es aus, indem Sie z.B. die Blinkrate verstellen (hier können Sie auch die +/– Tasten verwenden) und mit der rechten Maustaste auf "Abbruch" klicken.

Wird der HILFE-Knopf gedrückt, so wird help_settings aufgerufen. Anschließend wird der HILFE-Knopf, der ja selektiert war, wieder zurückgesetzt (undo_state) und der Knopf über draw_object neu gezeichnet.

LOCAL BOOLEAN key_settings (WINDOWP window, MKINFO *mk)

Diese Funktion wird vom Modul WINDOWS aufgerufen, wenn eine Taste gedrückt wurde. Sie muß hier immer FALSE liefern. Das edit_obj in der WINDOW-Struktur enthält die Objektnummer an die das Zeichen gegangen ist. In unserem Fall gibt es nur ein Editobjekt (SBLINK). Wir holen uns den Zeiger auf den Text des Objekts. Ist der Text leer, so soll der OK-Knopf grau werden, wenn er es nicht schon ist. Ist der Text nicht leer, so soll der OK-Knopf schwarz dargestellt werden, um zu signalisieren, daß der Dialogzustand gültig ist.

Diese Kontextsensitivität hilft dem Benutzer ungemein. Falsche Eingaben können schon im Vorfeld verhindert werden. So werden Fehlermeldungen nahezu überflüssig. Die Meldung "Undefinierte Blinkrate", die erscheinen müßte, wenn OK gedrückt wird und das Feld Blinkrate leer war, kann entfallen. Durch die Visualisierung (OK-Knopf grau) erhält der Benutzer außerdem sofort ein Feedback für seine Aktionen. Das Datenbanksystem "PHOENIX", welches bei Application Systems Heidelberg vertrieben wird und ebenfalls von uns entwickelt wurde, macht regen Gebrauch von kontextsensitiven und nicht-modalen Dialogfenstern.

Nur wenn sich der Zustand des OK-Knopfes ändert, muß dieser neu gezeichnet werden, da es sonst zu einem ständigen Flackern bei jeder Taste kommen würde. Dieser Umstand wird durch die elegante Formulierung

```
if ((*p == EOS) == ! is_state (settings, SOK, DISABLED))
```

herbeigeführt. Die erste Bedingung gibt immer an, wann der Knopf ungültig ist. Die zweite Bedingung gibt an, ob der Knopf gerade gültig ist (! DISABLED). Ist z.B. die Blinkrate leer und der Knopf "schwarz", so ergibt dies TRUE == TRUE. Genau dann ist es nötig, den Zustand des Knopfes zu ändern (flip_state) und diesen neu zu zeichnen.

Das gleiche gilt für "Blinkrate nicht leer" und "Knopf grau". Dann ist die Bedingung FALSE == FALSE ebenfalls erfüllt und der Zustand muß geändert werden. Voraussetzung für das Funktionieren des "flip_state"-Trickes ist, daß der Knopf vor dem Start des Dialoges korrekt eingestellt ist. Solch eine korrekte Einstellung sollte am Ende von set_... gemacht werden (in unserem Fall set_settings).

Hinweis: Die Funktionen key_ und click_ werden vom Modul WINDOWS indirekt angesprungen und erhalten zwei Paramter (window, mk). Oft ist es nötig, in diesen beiden Funktionen weitere Informationen über das Dialogfenster zur Verfügung zu haben. Was "normalen" Fenstern recht ist, sollte Dialogfenstern billig sein. Auch hier ist es möglich, in das Feld window->special z.B. einen Zeiger auf eine Struktur einzuhängen, die weitere Informationen beinhaltet. Dieser Zeiger ist dann in den key- und click-Funktionen verfügbar.

Beispiel:

```
typedef struct
{
  WORD exit_obj;
  WORD blinkrate;
} SET_SPEC;

SET_SPEC set_spec;

...
  window->click    = click_settings;
  window->key      = key_settings;
  window->showhelp = help_settings;
  window->special  = (LONG)&set_spec;
...
```

Und in click_... heißt es dann:

```
SET_SPEC *set_spec;

set_spec = (SET_SPEC *)window->special;
```

LOCAL BOOLEAN help_settings (WINDOWP window, WORD icon)

In der Help-Funktion wird nach einem vorhandenen Hilfe-Fenster gesucht. Ist keines da, so wird der Objektbaum auf die rechte untere Ecke des Bildschirmes gelegt. Dort stört das Hilfe-Fenster am wenigsten. Dann wird der Dialog kreiert (crt_dialog).

Bei erfolgreichem Erzeugen wird, wenn das Fenster nicht offen war, der Arbeits- sowie der Scrollbereich des Hilfe-Fensters ebenfalls auf den rechten unteren Bereich des Desktops gesetzt. Würde man das nicht tun, so würde das Hilfe-Fenster, wenn es einmal geschlossen wurde, immer an der alten Position erscheinen. Das soll beim Hilfe-Fenster aber gerade nicht passieren.

# Programmierung von Listboxen

Das Modul Windows wurde um die Darstellung von Listboxen erweitert. Eine Listbox besteht aus mehreren Objekten. Das Wurzelobjekt (ROOT) ist eine Box, die alle Komponenten einer Listbox enthält. Das Objekt ITEMS enthält alle Textzeilen einer Listbox. Das Objekt PARENT ist eine Box, die die Elemente der Schieber enthält. Er wird SLIDER genannt, die beiden Pfeile heißen UP und DOWN.

Im SCRAP-Programm wurden zwei Listboxen implementiert, um Zeichensätze bzw. Punktgrößen auszuwählen. Die Objekte der Font-Listbox beginnen mit SF... (SelFont), die der Punktgrößen mit SS (SelSize). Beispiel für die Font-Listbox:

```
#define SFROOT    3       /* BOX in tree SELFONT */
#define SFITEMS   4       /* BOX in tree SELFONT */
#define SFPARENT  12      /* BOX in tree SELFONT */
#define SFSLIDER  13      /* BOX in tree SELFONT */
#define SFUP      14      /* BOXCHAR in tree SELFONT */
#define SFDOWN    15      /* BOXCHAR in tree SELFONT */
```

Um weitere Listboxen in anderen Dialogboxen zu benutzen, kopieren Sie sich eine der beiden Listboxen in Ihre Dialogbox hinein. Um eine Listbox mit der Maus greifen zu können, gehen Sie im Resource Construction Set wie folgt vor: Drücken Sie die Control-Taste und schieben den gepunkteten Bereich des Sliders der gewünschten Listbox um ein Zeichen nach links (Durch das Drücken der Control-Taste erreichen Sie das Objekt SFPARENT). Darunter kommt das Wurzelobjekt (SFROOT) der Listbox zum Vorschein. Diese Objekt können Sie greifen und verschieben oder kopieren. Es beinhaltet alle Objekte der Listbox. Schieben Sie zum Schluß den gepunkteten Bereich des Sliders wieder in die alte Position.

Das Objekt SFITEMS können Sie selektieren, wenn Sie die Control-Taste betätigen und auf irgendeine Textzeile in der Listbox klicken.

Achtung! Sortieren Sie die Listbox nie, da sonst die benötigte Reihenfolge der Objekte nicht eingehalten wird!

Nach dem Kopieren von Listboxen geben Sie den 6 o.g. Objekten bitte wieder neue Namen. Die benötigen Sie für die Initialisierung der Listbox.

Die Datenstruktur LISTBOX (WINDOWS.H):

```
#define LIST_INIT   0x01          /* Initialisiere Listbox */
#define LIST_DRAW   0x02          /* Zeichne Listbox */
#define LIST_CLICK  0x04          /* Klick in Listbox */

typedef struct
{
  WINDOWP window;        /* Fenster der Listbox */
  OBJECT  *tree;         /* Objektbaum der Listbox */
  VOID    *itemlist;     /* Liste der Structs aller Einträge */
  SIZE_T  itemsize;      /* Größe eines Elements */
  BOOLEAN indirect;      /* Elemente sind Zeiger auf Zeichenketten */
  WORD    num_items;     /* Anzahl verfügbarer Einträge */
  WORD    vis_items;     /* Anzahl sichtbarer Einträge  */
  WORD    width;         /* Breite eines Eintrags */
```

```
  WORD    first_item;   /* Erster Eintrag in Listbox */
  WORD    active;       /* Aktiver Eintrag in Listbox */
  UWORD   sel_state;    /* Status, mit dem selektiert wird */
  WORD    root;         /* Objektnummer der Listbox */
  WORD    items;        /* Objektnummer der Box mit Einträgen */
  WORD    up;           /* Objektnummer des Hoch-Pfeils */
  WORD    down;         /* Objektnummer des Unten-Pfeils */
  WORD    parent;       /* Objnummer des Elternteils des Schiebers */
  WORD    slider;       /* Objektnummer des Schiebers */
} LISTBOX;
```

Bemerkungen:

Das Feld "itemlist" enthält die Texte aller Einträge. Die Texte können auch aus Strukturen bestehen, die weitere Informationen neben den Texten enthalten können. Im Font-Beispiel sind die Texte in "fnames" direkt gespeichert. Der zugehörige Speicher wird dynamisch alloziert und freigegeben, wenn das Dialogfenster wieder geschlossen wird. Die Texte in "itemlist" können breiter sein als der Platz, der in der Dialogbox reserviert ist. Die Zeichenketten werden vor der Ausgabe entsprechend gekürzt.

Beispiel für eine Struktur:

```
typedef struct
{
  LONG Nummer;
  BYTE Name [40];
  BYTE Ort [20];
} KUNDE;

KUNDE kunden [100];
```

Die Initialisierung für eine Namensliste muß wie folgt aussehen:

```
  fnames.itemlist  = kunden [0].Name;
  fnames.itemsize  = sizeof (KUNDE);
  fnames.indirect  = FALSE;
  fnames.num_items = 100;
```

Sind nicht die Namen, sondern nur Zeiger auf die Namen in der Struktur abgelegt, so muß das Flag "indirect" auf TRUE gesetzt werden. Beispiel:

```
typedef struct
{
  LONG Nummer;
  BYTE *Name;
  BYTE *Ort;
} KUNDE;

KUNDE kunden [100];

  ...
  fnames.indirect = TRUE;
```

Das Feld "sel_state" enthält den Status, der benutzt werden soll, wenn eine Zeile selektiert wird. Man wird CHECKED oder SELECTED benutzen. CHECKED

zeichnet ein Häkchen vor das selektierte Objekt (Namen sollten aus Platzgründen mit zwei Leerzeichen beginnen). SELECTED invertiert die aktuell angewählte Zeile.

Initialisierung für das FONT-Beispiel: (siehe MENU.C)

```
wnames = selfont [SFITEMS].ob_width / gl_wbox;
size   = wnames + 1;     /* EOS-Zeichen */
if (fnames == NULL)
  fnames = mem_alloc ((LONG)num_fonts * size);

lnames.window     = search_window (CLASS_DIALOG,
                                   SRCH_ANY, SELFONT);
lnames.tree       = selfont;
lnames.itemlist   = fnames;
lnames.itemsize   = size;
lnames.indirect   = FALSE;
lnames.num_items  = num_fonts;
lnames.first_item = (inx >= nlines) ? inx : 0;
lnames.active     = inx;
lnames.sel_state  = CHECKED;
lnames.root       = SFROOT;
lnames.items      = SFITEMS;
lnames.up         = SFUP;
lnames.down       = SFDOWN;
lnames.parent     = SFPARENT;
lnames.slider     = SFSLIDER;

if (fnames != NULL) set_lnames (num_fonts);
listbox (&lnames, LIST_INIT, NULL);
```

Ist die Struktur belegt, so ruft man listbox mit dem Befehl LIST_INIT auf. Der 3. Parameter ist ein Zeiger auf die Struktur MKINFO und wird nur bei LIST_CLICK benötigt. In obigem Beispiel setzt die Funktion set_lnames noch die Namen der Fonts in das Feld "itemlist".

Nachdem die Listbox initialisiert ist, muß sie auf Mausaktionen reagieren. In der Funktion click_font wird dies wie folgt erreicht:

```
if ((SFROOT <= window->exit_obj) &&
    (window->exit_obj <= SFDOWN))
{
  active = lnames.active;
  dclick = listbox (&lnames, LIST_CLICK, mk);
  set_fonttable (font_table [lnames.active]);

  if (active != lnames.active)
  {
    lsizes.first_item = 0;
    lsizes.active     = 0;
    sel_point         = point_table [0];
    num_points        = 0;

    while (point_table [num_points] != 0) num_points++;
    set_lsizes (num_points);
    listbox (&lsizes, LIST_INIT | LIST_DRAW, mk);
    draw_font (window);
  } /* if */
} /* if */
```

Zunächst wird getestet, ob sich das exit_obj im Bereich der Fontliste befindet (SFROOT bis SFDOWN). Dann merken wir uns den alten selektierten (aktiven)

Wert, um später testen zu können, ob der Benutzer ein anderes als das schon selektierte Objekt angewählt hat.

Jetzt wird die Funktion listbox aufgerufen. Der Befehl ist LIST_CLICK mit den aktuellen Mausinformationen (mk). Man erhält zurück, ob ein Doppelklick auf ein selektiertes Objekt angewendet wurde. Die Zeichensatzgrößen für den neuen selektierten Font werden berechnet.

Wurde ein neuer Font gewählt (ative != lnames.active), so wird die Listbox "lsizes", die ja die Punktgrößen enthält mit den Größen des neuen Fonts initialisiert, die Texte gesetzt (set_lsizes) und die Listbox sowohl initialisiert, als auch neu gezeichnet (LIST_INIT | LIST_DRAW). Der Beispieltext wird ebenfalls neu gesetzt (draw_font).

Schauen Sie sich bitte die Funktion draw_font genauer an, falls Sie in Dialogboxen etwas zeichnen möchten, was Sie nicht in der Resource-Datei anlegen können. Sie müssen dann Ihre Zeichenfunktion in die Fensterstruktur der Dialogbox einhängen. In mselfont wird dies mit

```
window->draw = draw_font
```

realisiert. In der Funktion draw_font darf das Clipping nur gesetzt werden, wenn es sich um das oberste Fenster handelt, sonst wird das Clipping ja durch das redraw vom Window-Manager automatisch gesetzt. Man muß es setzen, wenn das Fenster oben ist, da nach einem Scrollen oder Klicken in eine Listbox das Clipping von der Listbox-Funktion verstellt wird.

# Zusatz zur Bedienung von SCRAP

Die Menüleiste enthält jetzt unter dem Menü "Optionen" die Menüpunkte

Schriften laden
Schriften freigeben
Schriftauswahl...

Die Texte wurden komplett ins Deutsche übersetzt (ehemals Fonts laden usw.).

Fall Sie GDOS installiert haben, sollten Sie Schriften laden, um besser mit dem neuen Dialogfenster "Schriftauswahl" experimentieren zu können.

Das Dialogfenster "Schriftauswahl" ist ein sogenannter nicht-modaler Dialog. Nach der Anwahl des Menüpunktes "Schriftauswahl..." erscheint das Fenster in der Mitte des Bildschirms. Dort können Sie mit der Maus eine neue Schrift oder eine andere Schriftgröße auswählen. Ein Beispieltext zeigt sofort die Auswahl an.

Führt man einen Doppelklick auf eine Schrift oder eine Punktgröße aus, so wird das Dialogfenster mit "OK" verlassen.

Die ausgewählte Schrift wird benutzt, um das nächste noch nicht geöffnete "Power"-Fenster damit anzuzeigen. Ein geöffnetes "Power"-Fenster behält die Schrift, mit der es zum erstenmal geöffnet wurde, bei.

Da die Schriftauswahl nicht-modal ist, kann man mit der rechten Maustaste die Knöpfe drücken, ohne das Fenster zu schließen. Drückt man also mit der rechten Maustaste auf "OK", so wird die gewählte Schrift und Größe übernommen und für das nächste "Power"-Fenster benutzt. Drückt man mit der rechten Maustaste auf "Abbruch", so wird die letzte Einstellung zurückgeholt. Ein Klicken mit der rechten Maustaste auf die Knöpfe eines nicht-modalen Dialogfensters bewirkt also das gleiche wie mit der linken Taste, ohne jedoch den Dialog zu beenden.

Experimentieren Sie doch damit, daß Sie in der Funktion mselfont im Modul MENU.C statt WI_MODELESS ein WI_MODAL schreiben und die Auswirkungen beobachten.

Die ausgewählte Schrift, sowie die Schriftgröße werden im special-Wert der "Power"-Window-Struktur im zweiten sowie dritten Byte abgelegt.

# Stichwortverzeichnis


Accessories 12, 15, 96, 119, 183, 303, 335, 341, 365, 383, 402, 434, 447, 456, 458, 462, 485, 490
ac_close 123
ac_open 123
addr_in 104
addr_out 104
AES 18, 25, 42, 80, 96, 162
AES-Binding 104
AES.H 331, 407, 496
akustischer Abbildspeicher 292
Alcyon C 13, 379
Alertbox 141
Alternate-Taste 146
ANSI-C 14
ANSI-C-Compiler 339, 343, 361, 383
ANSI-Standard 211
any_open 430
appl_bvset 117, 119
appl_exit 53, 117
appl_find 117f.
Application-Library 99, 116
Applikation 96
appl_init 117
appl_read 117f.
appl_tplay 117f.
appl_trecord 117f.
appl_write 117f.
appl_yield 98, 117, 119
Arbeitsgerät 51, 57
Arbeitsstation 28, 32, 39, 53, 59, 61
array2rect 417
array2xywh 417
arrow-mouse 403, 408
arrow_window 437
ASCII-Code 120
ASCII-Dateien 14
aspect ratio 59
ASSIGN.SYS 19, 27, 29, 31ff., 35, 39, 42, 52, 57, 62, 95
Assoziationen 294f.
Atari 11
Atari ST 12f., 15, 18f., 21, 23, 33, 41, 56
Atari TT 12, 41
Attribute 28, 64
Attribute-Funktionen 29, 67
Aufmerksamkeit 313
Ausgabedatei 183
Ausgabefunktionen 29, 63
Ausgabegerät 39, 55, 57, 59

background 411
benutzerdefinierte Objekte 132, 286
Betriebssysteme 215
Bezier-Funktionen 186
Bildschirm 25, 28, 32, 43, 55, 59, 61ff., 74
Bildschirmauflösung 12, 58
Bildschirmfenster 24, 63
Bildschirmmanager 97, 121, 126, 153
Bildschirmspeicher 77
Bildschirmtreiber 34
Bindings 40, 42, 51
BIOS 18, 23
Bit-Image 25, 79
Bit-Image-Datei 27, 82, 183, 330, 333, 335, 340, 355
Bit-Image-Files 14
Bit-Image-Format 198
blink 414
busy_mouse 403, 408

CALCLOCK 61
CALCLOCK.ACC 123
click_window 441, 453
Clipboard 147, 306, 329ff., 371, 405
CLIPBRD 147, 354, 370, 373ff., 423, 430, 441, 445, 464, 482
clipping 63, 114
close_all 434
close_dial 412f.
close_top 434
close_vwork 406
close_window 402, 433
close_work 407
clr_area 446
clr_bottom 446
clr_left 446
clr_right 446
clr_scroll 446
clr_top 446
clr_work 446
Compiler 215
contrl 39, 42, 51
control 104
Control-Taste 146
Copy 147
create_windows 432
crt_clipbrd 466f.
crt_desktop 462f.
crt_disk 468
crt_edit 475
crt_graf 478
crt_image 472
crt_meta 473
crt_module 481, 484
crt_power 477
crt_printer 469
crt_trash 471
Cut 147

Dateiauswahl-Box 304f., 405, 427
Datenaustausch 328, 333
Datentypen 210
DCHECKBOX 133
delete_window 432, 434
Deskaccessories 303, 373, 402
Desktop 18, 20, 43, 51f., 55, 354, 373ff., 423, 441, 445, 452, 460
Desktop-Fenster 109, 150
DESKTOP.INF 76, 383
DHEADER 133
Dialogbox 111, 115, 309, 312f., 318f., 321ff., 364, 412
Dialog-Funktionen 411
Digital Research 288
Digital Research C 13, 222, 379, 523
DISK 354, 468
Dispatcher 97
do_flags 409
do_state 408
drag_boxes 444
drag_to_window 440
DRAW3D 132
draw_func 461
draw_key 461
draw_mbar 447
draw_mtitle 447
draw_object 444
draw_window 434
DRBUTTON 133
Drop-Down-Menü 115
Drucker 19, 25, 28, 32, 43, 55, 59, 61, 63, 69f., 85ff.
Druckerspooler 123
DUMPGEM 186
Dutch 33

EDIT 355, 368, 370f., 374, 376, 402, 475
Eingabe-Funktionen 30, 80
Entwicklungsumgebungen 209
environment 159
Ereignisse 99, 110, 120
Erkennungssystem 290f., 293
error 420
Error-Box 142
Escape-Funktionen 30, 42, 84
EVENT 353ff., 368, 399, 430, 436f., 442, 448, 451, 459, 479, 482
Event-Library 99, 119
evnt_button 119, 121
evnt_dclick 119, 125
evnt_event 451
evnt_keybd 119f.
evnt_mesag 119, 121
evnt_mouse 119, 121
evnt_multi 119f., 125, 368, 444, 451f., 482
evnt_timer 119, 125
Exponentialgesetz der Übung 312
EXPORT.H 397
Extended-Graphics-Library 103, 161

Farben 58, 69, 76, 199
Farbpalette 200
Fehleraufhebung 325
Fehlerbehandlung 324, 419
Fehlermeldung 115
Fehlervermeidung 324
Fenster 17, 102, 112, 150
file__exist 426
File-Selector-Library 102, 149
file__split 425
find__desk 460
find__flags 410
find__state 409
find__top 430
find__window 430
FlexOS 11, 13, 22, 215, 232, 301, 305, 308, 317, 378f., 383, 388ff., 395, 398, 488, 535
flip__flags 410
flip__state 409
form__alert 140f., 143
Formate 183
form__button 140, 144
form__center 111, 140, 143
form__dial 111f., 140, 142
form__do 111, 140, 142, 413
form__error 140, 142f.
form__keybd 140, 144
Form-Library 100, 140
Formular 140
fsel__exinput 149f.
fsel__input 149
full__window 439
Fundamentalprinzip der Aufgabenanalyse 300

GDOS 19, 21, 25ff., 31f., 42, 56, 97, 284
GDOS.PRG 27
GDP 42
GEM 5, 11f., 14, 17ff., 22, 301ff., 311, 318fzf., 322, 335, 341, 402, 451
GEM 2.x 329ff., 334, 374
GEM/3 11, 21, 29, 35, 39, 42f., 46, 47, 55f., 59f., 62, 64ff., 73, 82, 86ff., 94f., 116, 128, 143, 329ff., 354, 405, 464f.
GEM 3.X 147
GEMAIN 353, 355, 397, 455
GEM-Desktop 157
GEMDOS 13, 18, 23, 52, 215, 221, 333, 379, 383, 387, 395f., 399, 488, 523
GEMFILE.GEM 53, 91
GEM-Output 53
GEM-Versionen 215
Geräteformat 77
Gerätekennung 28, 31, 39, 42
gerätespezifisches Format 30, 74, 79
Gerätetreiber 27, 35
Gesetz der Übung 296
Gesetz von Fitt 309
get__border 444
get__clipxywh 465
get__dxywh 460
get__path 426
get__ptext 410
GLOBAL 104, 353, 400, 455
GLOBAL.H 371
GOMS-Modell 314
GRAF 355, 374, 478
graf__dragbox 144, 445
graf__growbox 144f., 161, 419, 444
graf__handle 34, 52, 108, 144, 146, 401
Grafiktablett 25, 28, 32
grafische Benutzeroberfläche 5
graf__mbox 144f., 210
graf__mkstate 120, 144, 146
graf__mouse 144, 146
graf__movebox 145, 210
graf__rubberbox 144
graf__rubbox 144
graf__shrinkbox 144f., 161, 419, 444
graf__slidebox 144f.
graf__watchbox 144f.
Graphics-Library 101, 144
GRC 14, 163
growbox 419, 439
G__USERDEF 132

handle 28
help__clipbrd 467
help__desktop 464
help__disk 469
help__edit 476
help__graf 480
help-image 473
help__meta 474
help__module 483
help__power 478
help__printer 470
help__trash 471
Hick's Gesetz 296

hide_mouse 403, 408
High C 236, 388, 535
Hilfesystem 325
hndl_button 453
hndl_dial 413
hndl_events 452
hndl_keybd 452
hndl_ml 453
hndl_menu 459
hndl_mesag 454
hndl_timer 454
Hochformat 87
HOTCLOSE 152
h_slider 437

IBM 11
IBM-PC 13, 15, 17, 20f., 23, 32, 41
Icon 17, 79
icons_clipbrd 466, 481
icons_module 481
Icon-Technik 306
IMAGE 354, 472
IMPORT.H 391
info_clipbrd 467
info_desktop 463
info_disk 468
info_edit 476
info_graf 479
info_image 473
info_meta 474
info_module 483
info_power 477
info_printer 470
Information Hiding 341
info_trash 471
init_clipbrd 468
init_desktop 464
init_disk 469
init_edit 476
INITERM 354f., 454
init_global 427
init_graf 480
init_image 473
init_initerm 404, 454
init_menu 459
init_meta 475
init_module 484
init_power 478
init_printer 470
init_resource 456
init_trash 472
init_windows 432, 455
inside 417
Interferenz 295
Interferenz im Kurzzeitgedächtnis 298
Interferenz im Langzeitgedächtnis 298
intin 39, 42, 51, 91, 104
intout 39, 42, 51, 104
is_menu_key 415, 449, 453
is_top 430

Kamera 19, 25, 28, 32, 61
key_all 442, 453
Keystroke-Level-Modell 314
key_window 442
Klemmbrett 14, 147, 306, 329, 339
Kommandozeile 158
Kontrollfeld 76
Kontroll-Funktion 28, 57
Koordinatensystem 26, 51ff.
Kurzzeitgedächtnis 291ff., 297ff., 302, 311, 314, 318, 323

Langzeitgedächtnis 291, 293ff., 297, 299, 325
Laser C 224, 380, 387, 524
last_mouse 407
Lattice C 13, 226, 381, 526
Lernprinzip 296

Macintosh 288, 311, 322, 325, 328
make 14, 224
Mark Williams C 13, 228, 382, 399, 527
maßstabsgerechte Zeichnung 59
Maus 25
Mausform 146
Mausfunktionen 407
Mausknöpfe 121
Megamax Laser C 13
mem_alloc 420
mem_avail 421
mem_free 420
mem_lmove 422
mem_lset 421
mem_move 421
Memory Form Definition Block (MFDB) 73
mem_set 421
Mengenfunktionen 422
Mensch-Maschine-Schnittstelle 289
Menü 294, 302f., 312, 316ff., 354f.,

370ff., 375ff., 412f., 415
423, 429ff., 447, 458
menu__bar 126, 183
menu__center 111
menu__click 126, 128
menu__icheck 126
menu__iecheck 126
menu__ienable 112, 126f.
menu__key 449
Menu-Library 99, 126
menu__manager 447
menu__normal 447
Menüpunkte 303, 315 f.
menu__register 126f., 402
menu__text 126
menu__tnormal 112, 126f., 447
menu__unregister 126f.
Menüzeile 109, 111, 126
META 355, 473
Metadatei 19, 25, 27f., 28, 32, 39, 43, 51ff., 59, 61, 70, 90, 96, 148, 183, 186, 198, 330, 333, 335, 340, 355
Metadatei-Format 25
Metadatei-Treiber 33
Metaware High C 13
Microsoft C 13, 232, 384, 530
mn__selected 122
Modul 341f., 344, 375, 390
MODULE 480
Modus 317f., 320
Motiv 5
motorische Systeme 290, 292
Motorsystem 293
move__window 440
MS-DOS 13, 23, 42, 51, 56, 78, 215, 232, 301, 317, 378, 383f., 395f., 399, 405, 488, 530
MS-Windows 18
Multitasking 11, 15, 22, 319
Multitasking-Betriebssystem 21
Multiuser-Betriebssystem 21

Nachfragefunktionen 30, 81
Nachricht 110, 117, 121
NDC-Koordinatensystem 43
normalisierte Gerätekoordinaten 26
Normalized Device Coordinates NDC) 26
note 419
num__windows 431

ob__flags 131
objc__add 128, 138
objc__change 112, 128, 140
objc__delete 128, 138
objc__draw 110f., 128, 139
objc__edit 128, 139
objc__find 112, 128, 139
objc__offset 128, 139
objc__order 128, 139
objc__rect 411
Object-Library 99, 128
Objekt 130
Objektbaum 115, 128, 138
Objektfunktionen 408
ob__spec 131
ob__state 111f., 131, 140
ob__type 131
open__clipbrd 467
open__desktop 463
open__dial 412f.
open__disk 468
open__edit 475
open__graf 479
open__image 472
Open-Look 5
open__meta 474
open__module 483
open__power 477
open__printer 470
open__trash 471
open__vwork 406
open__window 433, 463, 467
open__work 407
Open-Workstation 29
OS/2 11, 22, 215, 383, 388
OUT-Datei 86, 335
OUT-Dateiformat 207
OUTPUT 19, 27, 54, 61, 86, 90, 123, 198
OUTPUT.APP 333ff.

Papiergröße 54
Parameter-Block 40f.
Parameterübergabe 39
PARMBLK 134
Paste 147
path__exist 426
pb__type 132
Piktogramm 17, 25, 78f., 109, 112
Pixelbreite 59
Pixelgrafik 93, 198

Pixelgröße 54
Pixelhöhe 59
Plotter 19, 25, 28, 32, 43, 55, 61
Pop-Up-Menü 15, 17, 310f., 323f., 340, 353, 355, 403, 407, 412, 414f., 449, 477, 489
PORTAB.H 215, 345, 347, 491
Portabilität 209
POWER 355, 373f., 476, 482
Presentation Manager 5, 22
print_clipfiles 466
PRINTER 354, 469
Prinzip der spezifischen Verschlüsselung 295, 299
Prinzip des Problemraums 300, 313
Programmer's Toolkit 5f., 384
Prototyp 214, 343, 350, 388f.
Prozeß 98, 117, 119, 121
Prozessoren 215
ptsin 39, 51, 91
ptsout 39, 42, 51
Pull-Down-Menü 17, 25
Punkteregel 302

Querformat 87

Raster Coordinates (RC) 26
Rasterfunktionen 30, 73
Rastersystem 43
Rationalitätsprinzip 300, 314
rc_copy 416
rc_equal 416
rc_intersect 416
RCM 163, 175, 353f., 451
RCS 115
rc_union 416
Rechteckfunktionen 416
Rechtecklisten 113
rect2array 417
rect2xywh 418
redraw_window 435
RESOURCE 354, 406, 411, 420, 456
Resource-Construction-Set 109, 115, 128, 155
Resource-Datei 78, 108, 162
Resource-Library 103, 155
Resourcen 162
rsc_create 456
rsrc_free 155
rsrc_gaddr 109, 155f.
rsrc_load 109, 116, 155, 175, 428, 457
rsrc_obfix 155, 157
rsrc_saddr 155, 157

Scan-Code 120
Schieber 113
Schreibmodus 65, 67, 71
SCRAP 6
scrap_clear 465f.
Scrap-Library 101, 146
scrap_read 465
scrap_write 465
scroll_area 445
scrollen 79
scrolling 73
scroll_window 436
scrp_clear 149, 331
scrp_read 148, 331f.
scrp_write 148, 331ff., 465
search_window 429, 483
Seitenbreite 61, 198
Seitengröße 185
Seitenhöhe 61, 198
Seitenverhältnis 59
select_file 427
setall 423
setand 424
setcard 425
set_clip 402, 418
setclr 423
setcmp 425
setcpy 423
set_deskinfo 461
setexcl 424
set_func 461
setin 425
setincl 424
set_meminfo 462
set_mouse 407
setnot 423
setor 424
set_path 426
set_ptext 410
set_redraw 447
set_sliders 438
setxor 424
shel_envrn 157, 159
shel_find 157, 159
Shell-Library 103, 157
shel_rdef 157, 160, 334
shel_read 157f., 334, 399, 404, 428
shel_wdef 157, 160, 334

shel_write 157f., 333f., 455f.
Shift-Taste 146
SHOWGEM 240
show_mouse 403, 408
shrinkbox 419, 439
size_window 440
Slider 113
snap_window 438, 440
Speicherfunktionen 420
Standardformat 30, 74, 77, 79, 329f.
str_lower 422
str_upper 422
Style Guidelines 5f.
Swiss 33
Systemzeichensatz 401

term_clipbrd 468
term_disk 469
term_global 428
term_graf 480
term_image 473
term_interm 404, 455
term_menu 460
term_meta 475
term_module 484
term_power 478
term_printer 470
term_resource 458
term_trash 472
term_window 451
Textattribute 207
timer_all 443
timer_window 443
Toolkit 5
top_window 435, 463
TOS 18f., 23
TOUCHEXIT-Flag 142
trans_gimage 411
Transputer-Workstation 17
TRASH 354, 470
Treiber 31, 39
Turbo 56
Turbo C 13, 229, 235, 383, 386, 399, 529, 532
Typüberprüfung 211

Übung 312f.
unclick_window 442
Undo 306, 325
undo_flags 409
undo_state 409
Unentschlossenheit 315
UNIX 14, 17, 22
Unsicherheitsprinzip 296, 315
Unterscheidungsprinzip 295, 298
untop_window 435
updt_menu 458
USERBLK 134

v_alpha_text 84, 86
v_arc 64
v_bar 64
v_bit_image 84, 93, 186
v_cellarray 64
v_circle 64
v_clear_disp_list 84, 86
v_clrwk 61, 85
v_clsvwk 53, 62
v_clswk 54, 57, 61
v_contourfill 64
v_copies 84, 87
v_curdown 84
v_curhome 84
v_curleft 84
v_curright 84
v_curtext 84
v_curup 84
VDI 18, 24f., 27f., 39f., 42f., 51f., 95
VDI.H 335, 511
vdi_handle 55, 57, 62
v_dspcur 84f.
v_ecol 84
v_eeos 84
v_ellarc 64
v_ellipse 64
v_ellpie 64
v_enter_cur 84
v_etext 64f.
vex_butv 80f.
vex_curv 80f.
v_exit_cur 84f.
vex_motv 80f.
vex_timv 80f.
v_fillarea 64
v_form_adv 84
v_form_advance 85
v_get_driver_info 35
v_get_pixel 73
v_gtext 55, 64
v_hardcopy 84f.
v_hide_c 52, 80
visueller Abbildspeicher 291f.

v_justified 64, 66
v_marker 64
vm_coordinates 51
vm_coords 45, 54, 56, 90, 92
v_meta_extents 54, 56, 85, 90
vm_filename 85, 90f.
vm_pagesize 45, 51, 54, 56, 92
v_opnvwk 57, 62
v_opnwk 39, 52f., 55, 57, 62, 108, 146, 184
Vorhersehbarkeit 315, 323
v_orient 84, 87
v_output_window 84, 86
v_pieslice 64
v_pline 64
v_pmarker 64
vq_cellaray 81
vq_chcells 84
vq_color 81
vq_curaddress 84
vq_extnd 77, 79, 81f.
vqf_attributes 81
VQ_GDOS.S 222
vq_inmode 81
vq_key_s 80
vql_attributes 81
vqm_attributes 81
v_qmouse 80
vqp_error 89
vqp_filmname 84, 89
vqp_films 89
vqp_state 89
vq_scqan 84
vq_tabstatus 84
vqt_attributes 81
vq_tdimensions 84
vqt_extent 81f.
vqt_font_info 81
vqt_justified 81
vqt_name 81, 83
vqt_width 81
v_rbox 64
v_rmcur 84f.
vro_cpfm 82
vro_cpyfm 73, 78, 80
vrq_choice 80
vrq_locator 80
vrq_string 80
vrq_valuator 80
vr_recfl 64
vrt_cpyfm 73, 79
vr_trnfm 73, 77ff., 109
v_rvoff 84
v_rvon 84
vsc_expose 85, 89
vsc_form 80
vs_clip 57, 63
vs_color 68f.
vs_curaddress 84
vsf_color 65, 68
vsf_inferior 65
vsf_interior 68, 71
vsf_perimeter 65, 68, 73
vsf_style 65, 68, 71
vsf_updat 68
v_show_c 80
vsin_mode 80f.
vsl_color 64, 68
vsl_ends 64, 68
vsl_type 64, 68
vsl_udsty 68
vsl_width 64, 68
vsm_choice 80
vsm_color 64, 68
vsm_height 64, 68
vsm_locator 80
vsm_string 80
vsm_type 64, 68
vs_mute 84, 88
vsm_valuator 80
v_sound 84, 88, 419
vs_palette 84
vsp_film 84, 89
vsp_message 89
vsp_save 89
vsp_state 89
vst_alignment 65, 68, 70
vst_color 54, 68, 70
vst_effects 54, 65, 68, 70
vst_ex_load_fonts 57, 63
vst_font 54, 65, 68, 70, 83
vst_height 65, 68
vst_load_font 39
vst_load_fonts 53, 57, 70, 83
vst_point 55, 65, 68f.
vst_rotation 55, 65, 68
vst_unload_fonts 53f., 57, 63
vswr_mode 54, 65, 68, 71
vt_alignment 84
vt_axis 84
vt_origin 84
v_tray 84, 88

vt_resolution 84
v_updwk 53, 57, 61, 86
v_write_meta 85, 90
v_xbit_image 85, 93

Wahrnehmungssystem 290f., 293
WHITEBAK 132
wind_apfind 373, 440
wind_calc 150, 154
wind_close 114, 150, 152
wind_create 112, 150f., 433
wind_delete 114, 150, 152
wind_find 150, 153
wind_get 110, 150, 153
wind_new 150, 154
wind_open 113, 150, 152
Window-Library 102, 150
WINDOWS 353, 355, 357, 402, 428
WINDOWS.H 357, 362, 370, 372f.
wind_set 110, 113, 150, 153
wind_update 114, 150, 153
wm_arrowed 122
wm_closed 122
wm_fulled 122
wm_hslide 123
wm_moved 123
wm_redraw 122
wm_sized 123
wm_topped 122
wm_untopped 123
wm_vslide 123
Wordplus 86
workstation 28
writing mode 67

XBIOS 18, 23
XBRA-Protokoll 336
Xerox 289
X/GEM 11f., 21f., 39, 137, 209, 301, 303, 308, 319, 329f., 332, 365, 383, 388ff., 398ff., 412, 427, 451f., 455, 460
xgrf_2box 161
xgrf_stepcalc 161
X-Window 5, 17
xywh2array 417
xywh2rect 418

Zeichenkettenfunktionen 422
Zeichensatz 27, 31, 35, 39, 51, 53, 57, 62, 70, 83
Zeichenzelle 69
Zwischenablage 147, 306
Zykluszeit 291

# Die Autoren:

Wir wurden als eineiige Zwillinge am 4.6.1958 in Heidelberg geboren. 1980 begannen wir mit dem Studium der Informatik an der Universität Karlsruhe, das wir im Februar 1988 als Diplom-Informatiker abschlossen. Von 1986 bis Anfang/Mitte 1989 entwickelten wir das bekannte Datenbanksystem ADIMENS ST. Jetzt arbeiten wir bei der BASF AG als Informatiker. Unser Wunsch, komplexe Software für den Endanwender durch den Einsatz von grafischen Benutzeroberflächen bedienbar zu machen, ist dort auch eines unserer vorrangigen Ziele.

Während der letzten 2 Jahre entstand dann dieses Buch, das unser gesamtes GEM-Know-How beinhaltet. Es ist sowohl für den Anfänger, der noch gar keine Kenntnisse über GEM hat, als auch für den Fortgeschrittenen gedacht, der sein Wissen noch erweitern möchte. Der Programmierer findet hierin ein mächtiges Werkzeug zur Entwicklung portabler und komplexer GEM-Applikationen.

Günther Bengel

Hüthig

# Betriebssysteme

## Aufbau, Architektur und Realisierung

1990, 237 S., 62 Abb., kart.,
DM 58,—
ISBN 3-7785-1989-1

Ein Betriebssystem gehört zu jedem Rechner. Es verwaltet die Hardware und Software und vereinfacht die Bedienung und Anwendung des Rechners für den Benutzer.

Ausgehend von einer abstrakten Maschine bespricht der Autor zunächst einen Betriebssystemkern mit minimalem Funktionsumfang (Prozeßverwaltung, -umschaltung und -synchronisation, Ein-/Ausgabe, Zeitdienst). Darauf werden schrittweise und systematisch die verschiedenen weiteren Komponenten eines Betriebssystems (Scheduler, komfortablere Synchronisations- und Kommunikationsfunktionen, Hauptspeicher-, Geräte und Dateiverwaltung) aufgebaut.

Zur Beschreibung der Algorithmen, der Verdeutlichung der strukturellen Zusammenhänge zwischen den verschiedenen Komponenten und zur Abstraktion von Details dient die Programmiersprache Ada.

Die Darstellung orientiert sich nicht an einem speziellen konkreten Betriebssystem, sondern an dem allgemeinen Aufbau für die meisten Betriebssysteme, wobei jedoch diese Prinzipien durch Bezug auf konkrete Betriebssysteme (Unix, DOS, VAX-VMS) untermauert werden.

082643

Hüthig Buch Verlag
Im Weiher 10
6900 Heidelberg 1

Arnold Hickersberger

# Mit C zum Ziel

**Das C-Buch für Ein- und Umsteiger**

1989, 192 S., kart., DM 38,—
ISBN 3-7785-1555-1

Das Buch bietet eine Einführung in die Programmiersprache C. Der Sprachbeschreibung ist der Standard von Kerninghan und Ritchie zu Grunde gelegt. Schwerpunkte wurden in folgenden Bereichen gelegt: Datentypen, Speicherklassen, Operatoren, Kontrollstrukturen, Zeiger und Felder, Funktionen, Strukturen, Varianten und Bitfelder, Ein- und Ausgabe, Präprozessor. Die Beispiele sind nicht rechnerabhängig.

Das Buch wendet sich speziell an Leser mit Vorkenntnissen in PASCAL oder BASIC.

082641

Hüthig

Hüthig Buch Verlag
Im Weiher 10
6900 Heidelberg 1

Dieses Lehrbuch stellt Richtlinien für die Softwareentwicklung unter GEM auf. Sie gelten sowohl auf dem Atari ST als auch auf dem IBM-PC und dazu kompatiblen Rechnern sowie für alle Versionen von GEM, vom Ur-GEM über GEM/3 bis zum X/GEM.

Sie können dieses Werk als INSIDE GEM ansehen, das Softwareentwicklung unter GEM inklusive Theorie der Benutzeroberflächen vollständig beschreibt. Es entstand in Zusammenarbeit mit der Atari Computer GmbH und Digital Research.

Entwickler müssen mit diesem Tool nur noch einen minimalen Aufwand betreiben, um perfekte Software unter GEM zu erstellen. Sie müssen lediglich die Funktionalität Ihrer Applikation an das vorhandene Skelett-Programm SCRAP anhängen und die Regeln in Kapitel 5 beachten. Die komplexe Steuerung von Software unter einer graphischen Oberfläche wird Ihnen dabei komplett abgenommen.

Einige Highlights der Entwicklung: Es wurden Routinen implementiert, die z.B. Pop-Up-Menüs oder Menüzeilen in Fenstern möglich machen. Auch Accessories dürfen einen eigenen Desktop haben. Dieser wird dann in ein Fenster gelegt. Die entsprechenden Routinen werden jeweils genau erläutert.

In der Neuauflage wurden neben vielen Kleinigkeiten folgende Erweiterungen vorgenommen:

- Unterstützung beider Mausknöpfe
- modale und nicht-modale Dialogboxen in echten Fenstern
- erweiterte Alertboxen in echten Fenstern
- blinde Paßwort-Eingabe
- Unterstützung der TOS 1.4 File-Selector-Box
- Listboxen in Dialogboxen.

Mit der hier beschriebenen Methode können Sie GEM-Software so entwickeln, daß sie auf jedem Rechner (PC oder ST), mit jedem Betriebssystem, mit jeder Auflösung, mit jeder GEM-Version, als Programm oder Accessory mit jedem Compiler läuft.

ISBN 3-7785-2049-0

Hüthig