GAG-NEWS

Das Mitteilungsblatt der German Archimedes Group

Ausgabe 1 vom 1. April 92

Inhalt:

Herbert zur Nedden
Alte Landstraße 21
22962 Siek
☎ (0 41 07) 99 00

Olaf Krumnow
August-Bebel-Straße 102c
21029 Hamburg
☎ (0 40) 7 21 92 87

Preis: DM 2.10 zzgl. Porto Clubkonto: H. zur Nedden, Sonderkto. G, Nr. 6912 14-208, Postgiroamt Hamburg, BLZ 200 100 20

Olaf und Herbert machen einen GAG

Die German Archimedes Group (kurz **GAG**) ist ein deutschsprachiger Club für Archimedes-Fans. Das Hauptziel des Clubs ist die Unterstützung von denen, die einen Archimedes einsetzen. Dabei hoffen wir auf eine gute Zusammenarbeit mit den Mitgliedern und den Händlern. Die GAG ist kein eingetragener Verein, sondern wird von Olaf Krumnow und Herbert zur Nedden als Hobby, also nicht-kommerziell betrieben.

Du

Es stellt sich die Frage, wie man die Leser unserer Club-Zeitschrift, der **GAG-NEWS,** ansprechen soll. Es gibt dafür drei Alternativen: 'Man' ist uns zu unpersönlich und 'Sie' für eine solche Club-Zeitung auch nicht besonders passend. Daher werden wir die Artikel in der 'Du'-Form schreiben. Wir hoffen, daß Ihnen/Dir das so recht ist.

Wer sind Olaf und Herbert ?

Kennengelernt haben wir uns beim **MTX-User Club Deutschland**, einem Club, der sich um den MTX der englischen Firma Memotech drehte. Dieser Z80-Rechner hat mit dem Archimedes einiges gemein: beide kommen aus England, haben ein Basic mit eingebautem Assembler, sind Exoten, können hardwaremäßig erweitert werden und sind einfach GUT!

Im Laufe der Jahre wurde der MTX durch den Club erheblich verbessert und das Betriebssystem wuchs mit. Obwohl der MTX unter unserem CP/M 2.2-kompatiblen Betriebsystem läuft, welches auf ZCPR 3.3, einem erweiterten Public-Domain BDOS und einem selbstgeschriebenen BIOS basiert, ist er erheblich leistungsfähiger als man meinen könnte. Trotzdem stellten wir uns die Frage, ob wir die Z80 durch eine Z280 ersetzen oder uns einen neuen Rechner aussuchen sollten.

Zur Auswahl standen **Apple**, **Amiga**, **Atari** und **Archimedes** (daß alle mit einem 'A' beginnen ist Zufall). PCs oder dazu kompatible schieden aus - DoMessDos ist einfach kein Betriebssystem in unseren Augen. Klarerweise ist der Archimedes der interessanteste aus obiger Liste! Er hat ein sehr schönes Betriebssystem, welches an einigen Stellen sogar besser als das von Apple zu bedienen ist. Und das mit dem assemblerfähigen Basic und die Möglichkeiten der Hardware-Erweiterung stellte sicher, daß wir uns gegenüber dem MTX nicht verschlechtern würden.

Olaf hat sich vor über einem Jahr einen A410 gekauft, der bald mit 4MB Speicher und einer SCSI-Festplatte und mittlerweile mit dem ARM3 versehen wurde. Ich (Herbert) habe noch etwas gewartet, da bei mir das Geld zu knapp war. Es hat sich gelohnt: A5000 mit dem neuen RISC OS3 und dem ARM3. Die 40MB-Platte reicht sicherlich in vielen Fällen aus; da ich jedoch programmieren und außerdem den schnellen Rechner nicht mit einer relativ langsamen Festplatte ausbremsen wollte, spendierte ich mir eine etwas größere und schnellere Platte. Die 4MB Speicher ließen auch nicht lange auf sich warten.

Warum einen Club ?

Mit dem MTX User-Club Deutschland haben wir festgestellt, daß ein Club sehr viel Spaß machen kann und man dann auch nicht so alleine mit seiner Kiste ist. Es gibt zwar schon einige Clubs, aber halt nicht so einen, wie wir uns vorstellen.

Viele Clubs arbeiten mit Mitgliederbeiträgen und geben regelmäßig eine Club-Zeitung heraus. Ist viel zu veröffentlichen da, kann es sein, daß nicht alles in die Zeitung kommt, da sie sonst zu dick und damit zu teuer wird. Fehlen Artikel, dann haben die 'Macher' ein Problem.

Wie beim MTX-User Club Deutschland gehen wir den Weg, daß es keine festen Mitgliederbeiträge gibt. Du wirst zur Kasse gebeten wird, wenn wir Dir etwas schicken. Bei unserer Clubzeitung, der **GAG-NEWS**, funktioniert das so, daß alle interessierten Club-Mitglieder bei mir ein Guthaben haben sollten, von dem ich dann die Kosten für die **GAG-NEWS** abbuche. Deren Höhe hängt vom Umfang der Zeitung (und damit den Druckkosten) ab, liegt aber nie über DM 12.-. Haben wir besonders viele Beiträge, erscheint die **GAG-NEWS** öfter, anderenfalls kann es auch mal etwas länger dauern. Wenn Du die **GAG-NEWS** nach Erscheinen automatisch haben möchtest, mußt lediglich sicherstellen, daß Dein Guthaben bei mir nicht unter DM 12.- sinkt.

GAG-NEWS

In unserer Club-Zeitung sollen Artikel zu den Archimedes-Rechnern (Hard- und Software) erscheinen. Da Olaf und ich diese Artikel sicherlich nicht alle selbst schreiben können, sind alle Mitglieder - also auch Du - gebeten, uns Artikel oder Anregungen für solche zu schicken. Artikel, die wir von Mitgliedern erhalten, übernehmen wir unverändert (bis auf das Layout) in die **GAG-NEWS**, es sei denn, wir haben das Gefühl, daß sie fehlerhaft sind oder ein sehr ähnlicher gerade erschienen ist. In diesen Fällen melden wir uns auf jeden Fall beim Autor und reden darüber.

Weiterhin sollen in der **GAG-NEWS** auch andere Dinge erscheinen. Dazu gehören solche Artikel wie dieser, der sich mit der GAG selbst beschäftigt, aber auch Fragen und deren Antworten, Tips, Tricks, und Kleinanzeigen von Mitgliedern. (Größere Anzeigen werden wir auf Anfrage auch aufnehmen - bei diesen muß dann natürlich der Anbieter die Druckkosten tragen.)

Folgendes liegt uns sehr am Herzen: In unseren Augen gibt es keine 'zu banalen' Artikel und keine 'dummen' Fragen!

Besonders interessant für andere Mitglieder und uns sind vermutlich Besprechungen von Programmen. Es gibt mittlerweile eine Unmenge von Software für den Archimedes, die kaum einer übersehen kann. Hast Du z.B. ein noch so kleines Programm entdeckt, welches Du verwendest, weil es Dir hilft, schick uns doch ein paar Zeilen dazu.

Am Ende jedes Artikels findest Du den Autoren; bei Olaf und mir sind dies lediglich unsere Initialen OK bzw. HzN.

Wie schreibe ich für die GAG-NEWS

Wir haben uns vorgenommen, die **GAG-NEWS** mit **Impression** zu erstellen. An dieser Stelle vielen Dank an Rolf Herberg, der uns hierbei mit Tat und Software unterstützt: Er hat Impression samt dem zugehörigen Dongle schon eine ganze Weile.

Bitte schick uns alles, was wir in die **GAG-NEWS** aufnehmen sollen, auf Diskette. Texte können z.B mit !Edit oder einem anderen einfachen Editor erstellt werden, Bilder mit !Paint oder mit !Draw. Bei Verwendung von !Edit versuch bitte, <Return> nur am Absatzende einzugeben, auch, wenn der Umbruch, den !Edit dann am Bildschirm mitten im Wort macht, etwas unschön aussieht. Solltest Du eine Textverarbeitung verwenden, so schick uns bitte zur Sicherheit den Text zusätzlich als ASCII-Datei mit (das sollte die Textverarbeitung sicherlich ausgeben können) - wir kennen Impression noch nicht gut genug, um zu wissen, was damit alles fehlerfrei geladen werden kann. Selbstverständlich können Deine Beiträge auch als Impression-Dokument kommen.

Acorn hat ja leider noch keine deutschen Tastaturen oder Tastaturkappen. Daher ist es sicherlich etwas umständlich, mit deutschen Umlauten zu arbeiten. Trotzdem möchten wir Euch bitten, Umlaute zu verwenden, da sich das deutlich besser liest.

Wenn Du Impression hast und häufiger Beiträge schreiben magst, schicken wir Dir gerne unser Layout-Dokument für die **GAG-NEWS**.

Wir sind noch nicht sicher, in welcher Form wir **GAG-NEWS**-Artikel honorieren werden. Immerhin wollen wir mit Selbstkosten arbeiten. Die Diskette(n) erhältst Du selbstverständlich von uns mit der nächsten Sendung zurück.

GAG-SOFT

Olaf und ich haben vor, eine Software-Gruppe innerhalb der GAG auf die Beine zu stellen. Sobald wir hier Konkretes sagen können, schreiben wir auch darüber.

MP

Die Zeitschrift MP Mikroprozessortechnik aus dem Verlag Technik GmbH hat seit einiger Zeit in jeder Ausgabe Artikel über den Archimedes und will das auch weiterhin so halten. Olaf und ich können Euch die MP empfehlen - wer sonst außer uns schreibt schon über diesen Rechner? Aber auch die anderen Artikel können durchaus interessant sein. Wir haben vor, mit der MP-Redaktion zusammenzuarbeiten. Näheres dazu in der nächsten GAG-NEWS.

Telefon

Ihr könnt uns auch gerne anrufen, wenn Ihr Fragen o.ä. habt oder uns etwas berichten wollt. Unsere Telefonnummern stehen auf dem Deckblatt.

(HzN)

Multitasking-Programmierung

Wer, wie Olaf und ich, von einem Betriebssystem wie CP/M kommt, bei dem immer nur ein Programm läuft (es gibt zwar Tricks, mehrere Programme gleichzeitig im Speicher zu halten, aber laufen tut in der Regel nur eines, von Interrupt-Routinen und einem Druckerspooler mal abgesehen) und jetzt unter RISC OS programmieren möchte, muß umdenken. (Ähnlich geht es denen, die von MSDOS auf OS/2 umsteigen.)

Singletasking

CP/M ist wie MSDOS ein Singletask-Betriebssystem, d.h. daß das Programm, welches gerade läuft, sagt, was passieren soll. Nehme ich als Beispiel einen Texteditor, so sieht dessen Skelett etwa wie folgt aus:

```
Lies Zeichen von der Tastatur ein.
Ist es ein Befehl,
Dann führe ihn aus;
Anderenfalls übernimm das Zeichen in
     den Text.
```

Soll ein Text geladen werden, gibt der Anwender den entsprechenden Befehl ein. Tut der Anwender jedoch gar nichts, d.h. drückt auf keine Taste, dann steht der Rechner halt so in der Gegend herum.

Selbst wenn man eine Maus verwendet, bleibt es dabei, daß das Programm das Sagen hat und nicht das Betriebssystem.

Multitasking

Haben wir es mit einem Betriebssystem zu tun, welches Multitasking unterstützt, dann laufen mehrere Programme gleichzeitig. Das bedeutet genaugenommen, daß die Programme im Wechsel an die Reihe kommen. Damit dieser Wechsel möglich ist, müssen alle Programme regelmäßig die Kontrolle an das Betriebssystem abgeben, was ggf. auch mit Gewalt passiert.

Druckerspooler

Das einfachste Beispiel für Multitasking ist ein Drukkerspooler. Meist wird dieser so realisiert, daß wenn ein Programm auf eine Tastatureingabe wartet, jedoch der Anwender diesem Wunsch des Programmes noch nicht nachgekommen ist, ein Zeichen aus einem Puffer oder einer Datei ausgedruckt wird. Der Druckerspooler nutzt also die Zeit aus, in der der Rechner an sich nicht zu tun hat.

Timesharing (Preemptives Multitasking)

In einigen Fällen wird Multitasking erzwungen: Ein Programm wird durch das Betriebssystem einfach unterbrochen, damit andere an die Reihe kommen. Zu einem späteren Zeitpunkt 'darf' es dann an der unterbrochenen Stelle weitermachen.

Diese Art des Multitasking wird in der Regel bei Systemen eingesetzt, bei denen mehrere Anwender gleichzeitig arbeiten, also sog. Multiuser-Systemen. Hiermit wird erreicht, daß jeder Anwender seinen Anteil an Rechenleistung erhält.

Da das langsamste am Rechner in der Regel der Benutzer an seiner Tastatur ist, erfolgt beim Timesharing der Wechsel zwischen verschiedenen Anwendern nicht nur, wenn dessen Zeit abgelaufen ist, sondern auch dann, wenn das jeweilige Programm auf eine Tastatureingabe o.ä. wartet - der Rechner also eh für den Anwender im Moment nichts tut. Diese Technik, die auch der o.g. Druckerspooler nutzt, erhöht den Durchsatz des Systems ohne den Anwender zu beeinträchtigen.

Kooperatives Multitasking

Diese Art des Multitasking setzt, wie es der Name schon suggeriert, voraus, daß die Programme von sich aus die Kontrolle an das Betriebssystem abgeben.

Die meisten Programme geben ja laufend die Kontrolle ans Betriebssystem ab, da sie es für alles mögliche (Zeichen von der Tastatur lesen, Daten auf Diskette speichern, u.s.w.) aufrufen. Programme, die sich längere Zeit nur mit sich selbst beschäftigen, wie z.B. solche, die aufwendige Bilder berechnen, müssen ihre Arbeit in regelmäßgen Abständen freiwillig unterbrechen und die Kontrolle an das Betriebssystem abgeben.

RISC OS-Programme verhalten sich ürigens in der Regel genau so. Solche, die es nicht tun, verhindern, daß die anderen Programme zwischenzeitlich weiterlaufen. Dazu mehr weiter unten.

Fenster (Windows)

Heutzutage will ein Anwender häufig mehrere Programme gleichzeitig benutzen - mit gleichzeitig meine ich natürlich, daß sie alle im Speicher sind, und ich problemlos zwischen ihnen wechseln kann, ohne daß deswegen Programme beendet oder Daten gesichert werden müssen oder verloren gehen.

Um derartiges zu ermöglichen, wird in der Regel eine Benutzeroberfläche mit Fenstern (engl. Windows) verwendet. Jedes laufende Programm verwendet für seine Bildschirmausgaben ein Fenster und nicht den gesamten Bildschirm. Der Wechsel von einem zu einem anderen Programm erfolgt dann in der Regel, indem ich mit der Maus auf das Fenster des gewünschten Programmes gehe und es anklicke oder sonstwie anspreche.

Nachrichten (Messages)

In einer solchen Fenster-Umgebung ist es in der Regel auch erforderlich, daß sich Programme miteinander unterhalten können. Ein einfaches Beispiel dafür wäre der Fall, daß der Anwender das Icon einer Textdatei auf den Texteditor (unter RISC OS auf das Icon des Texteditors unten in der Icon-Leiste) zieht. Das gewünschte passiert: der Text wird in den Editor geladen und kann bearbeitet werden. Im Hingergrund passiert jedoch einiges: der Filer, d.h. das Programm, welches u.a. die Directories anzeigt erkennt, daß das Icon einer Textdatei verschoben wird. Sobald die Verschiebeoperation beendet ist, und der Filer erfahren hat, wohin 'sein' Icon geschoben wurde (in diesem Fall auf das Texteditor-Icon), schickt er eine Nachricht an den Empfänger (hier den Editor) mit der Aufforderung, den besagten Text zu laden.

Obiges Beispiel zeigt auch schon auf, wo die Tücke ist, an die man sich bei der Programmierung unter RISC OS gewöhnen muß: Der Texteditor ahnt nichts böses und plötzlich kommt die Aufforderung, einen Text zu laden. Und das kann passieren, obwohl er schon einen anderen Text geladen hat, und darauf wartet, daß der Anwender dort Text eingibt. Und statt ein Zeichen zu erhalten, muß er einen weiteren Text laden und zum Bearbeiten anbieten.

Beim Speichern ist das Spiel ggf. ein ähnliches. Der Anwender wählt die Option zum Sichern des Textes aus und erhält unter RISC OS in der die sog. Save-Box, ein Fenster, in dem er den Dateinamen eingeben, OK anklicken oder aber das Icon der Textdatei verschieben kann. Wird das Icon in eine Directory-Anzeige verschoben, so wird der Text dort gespei-chert. Dies funktioniert jedoch nur, da sich hier wieder der Editor und der Filer (erinnere, daß Directory-Anzeigen Fenster des Filers sind) unterhalten: Der Texteditor fragt den Filer, in welches Directory der Text gespeichert werden soll und sichert ihn anschließend.

Weiterhin kann es aber auch passieren, daß der Anwender im Taskmanager den Texteditor abbricht. In diesem Fall schickt der Taskmanager dem bis dato noch glücklichen Editor eine Nachricht mit der Aufforderung, daß er sich doch bitte verabschieden möge.

Kommen wir auf das o.g. Beispiel eines Texteditors zurück. Für ein Multitasking-Betriebssystem wie RISC OS sieht das Skelett so aus:

```
Warte auf die nächste Nachricht.
Ist es eine Tastatureingabe?
Dann falls es ein Befehl ist, führe ihn
    aus; sonst übernimm das Zeichen in
    den Text.
Anderenfalls führe den in der Nachricht
    enthaltenen Befehl aus.
```

Der Hauptunterschied zum zuerst gezeigten Editor-Skelett ist, daß der Editor vorher nicht 'weiß', ob nun eine Tastatureingabe oder etwas anderes kommt.

Single-Task im Multitasking

Eigentlich klingt das zwar paradox, aber wird gelegentlich gebraucht. Laufen mehrere Programme gleichzeitig, von denen man jedoch im Moment nur eines benutzen will, und dieses relativ viel Rechnezeit benötigt, dann kann man natürlich alle anderen Programme aus dem Speicher werfen. Üblicherweise gibt es jedoch einige Programme, die man nicht einfach beenden darf wie z.B. den Taskmanager oder den Filer. Dann ist es recht praktisch, wenn bestimmte Progamme die Möglichkeit bieten, die anderen nicht mehr oder nur noch kaum zum Zuge kommen zu lassen. Beispiele hierfür sind der PC-Emulator, der eben diese Option bietet oder Impression, welches sich beim Drucken fast der ganzen Rechnerleistung bemächtigt.

Eine andere Situation, in der abgesehen von einem Programm keine anderen mehr laufen, habe ich zum Glück dank RISC OS 3 nicht mehr kennenlernen müssen: Unter RISC OS 2 monopolisiert das Kopieren von Dateien oder das Formatieren von Disketten den Rechner - es sei denn, man verwendet dafür entsprechende Programme. Einige Hardware ist so

geartet, daß sie es nicht oder nur mit einigem Aufwand zuläßt, das sie nicht die 100%ige Aufmerksamkeit des Betriebssystems hat. In diesen Fällen hat sich das mit dem Multitasking natürlich auch erledigt.

RISC OS

Durch meine obigen Beispiele ist es sicherlich schon klar:

RISC OS ist ein kooperatives Multitasking-Betriebssystem mit Fenstern und Nachrichten.

Unter RISC OS gibt es ja nicht nur Tastatureingaben und Nachrichten anderer Programme. Da wären noch die Fenster mit ihren diversen 'Knöpfen' und die Maus. Je nachdem, was der Anwender mit seiner Maus wo macht, wird eine entsprechende Nachricht vom Betriebssystem an die betroffenen Programme geschickt. Sei es nun, daß der Anwender irgendein Text-Icon mit Inhalt gefüllt oder auch nur angeklickt hat, oder daß er sich mit den Scroll-Balken vergnügt hat.

Keine Angst, viele Aktionen, die man mit einem Fenster machen kann, wie es z.B. durch die Gegend zu ziehen, andere darüber zu packen oder darüberliegende wegzuziehen, die Größe zu verändern usw. brauchen das zugehörige Programm vielleicht gar nicht zu interessieren: Es ist beachtenswert, wieviel dieser Bildschirm- und Fensterverwaltungsaufgaben RISC OS für uns, sprich die Programme übernimmt. Erst wenn etwas für das Programm relevantes passiert oder RISC OS nicht in der Lage ist, den Inhalt des Fensters selbst zu aktualisieren, wird das Programm bemüht, d.h. erhält eine Nachricht. Weiterhin kann der Programmierer in Grenzen festlegen, was für das jeweilige Programm von Interesse ist. Viele Programme interessiert es wohl kaum, wenn der Mauszeiger über ihr Fenster huscht - andere wollen das wissen, um dessen Aussehen zu verändern.

Das Skelett eines RISC OS-Programmes, welches unter dem Wimp läuft, sprich fensterlt, ist:

```
Initialisierung mit Wimp_Initialize-
     Aufruf
Schleife mit Wimp_Poll-Aufruf (hole
     Nachricht) und der korrekten
     Behandlung der Nachricht.
Beenden mit Wimp_CloseDown-Aufruf.
```

Master und Slave

Es ist sicherlich dem einen oder der anderen aufgefallen: Programme, die in einer Singletasking-Umgebung wie CP/M laufen, bestimmen, solange sie laufen, was Sache ist. Sie beherrschen sozusagen den Rechner. Im Timesharing bestimmt das gerade aktive Programm i.a. darüber, was abläuft. Das Betriebssystem unterbricht es halt nur in gewissen Zeitabständen. Beim kooperativen Multitasking wie unter RISC OS hingegen, sieht die Welt anders aus (wenn die Programme anständig, d.h. Acorns Richtlinien entsprechend geschrieben sind): Das Betriebssystem hat die Kontrolle über den Rechner und die Programme müssen (mehr oder weniger) das tun, was ihnen vom Betriebssystem aufgetragen wird. Das ist halb so dramatisch, da Du als Anwender natürlich durch das, was Du über Tastatur und Maus eingibst, der eigentliche Herr im Hause bist.

Warum ??

Wozu eigentlich das Ganze mit dem Multitasking (mal von dem obligatorischen Druckerspooler abgesehen) und den Nachrichten und vor allem den anderen Programmiertechniken?

Das bringt eine Fenster-Benutzeroberfläche mit sich, wenn sie auch einigermaßen komfortabel sein soll. Will ich das Grafikprogramm starten können, indem ich einfach die zugehörige Datei anklicke, dann muß das Zeichenprogramm irgendwie gestartet und die Datei geladen werden. Das kann ich auch noch unter CP/M realisieren, wenn's auch etwas Arbeit macht (für meinen Memotech-Rechner haben wir so etwas ähnliches sogar programmiert). Was aber, wenn auch noch mal eben ein Text verändert und die Grafik dort hinein soll. Am besten wäre es doch, wenn ich die Grafik vom Zeichenprogramm aus in den Text 'schieben' könnte. Da wird es schon interessanter, da das Zeichenprogramm irgendwie mit dem Texteditor kommunizieren muß (sprich die beiden müssen Nachrichten untereinander austauschen). Und hat man schon einmal mit den Nachrichten angefangen, dann ist es auch nicht weiter wild, selbst den Teil des Betriebssystems, der die Disketten und Festplatten verwaltet, wie ein normales Programm laufen zu lassen, d.h. auch hier mit Nachrichten zu arbeiten. Wenn irgendein Fenster verschwindet, dann muß das darunterliegende ja auch irgendwie auf dem Bildschirm erscheinen - diesmal ist das Betriebssytem der Nachrichtenversender.

(HzN)

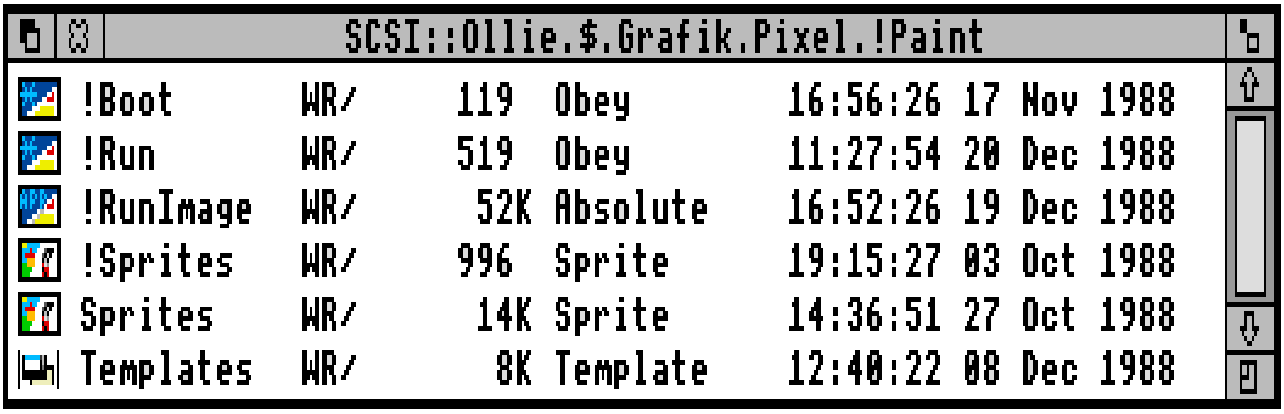
Aufbau einer Application

Neben einigen anderen Möglichkeiten, ist die Application die zentrale Art, Programme auf einem Speichermedium abzulegen. Eine Application ist genaugenommen ein Unterverzeichnis (Subdirectory), dessen Name mit einem Ausrufezeichen beginnt. Solche Directories werden vom Filer (das ist der Teil des Betriebssystems, der die Dateien verwaltet) besonders behandelt. Im folgenden will ich aufzeigen, welche Dateien in eine Application gehören, und was diese Dateien bewirken. Ich werde die Application **!Paint** für meine Beispiele heranziehen.

Das eigentliche Programm innerhalb einer Application heißt **!Runimage**, obwohl sich leider nicht alle Programmierer an diese Abmachung halten. Diese Datei enthält den auszuführenden Programmcode, und kann in einer beliebigen Programmiersprache geschrieben sein. Sehr häufig wird BASIC benutzt. Bei einem Doppelklick auf eine Application wird jedoch nicht die Datei **!Runimage** gestartet, sondern **!Run**. Diese Datei muß deswegen immer vorhanden sein. Ohne sie läßt sich eine Application nicht starten. Die Datei **!Run** kann von jedem beliebigen Typ sein, ist aber normalerweise vom Typ **Obey**. *[Eine* **Obey***-Datei ist, einfach gesagt, eine Sammlung von Kommandozeilen-Kommandos (CLI-Commands), die beim Starten dieser Datei nacheinander ausgeführt werden.]* **!Run** versichert sich aller Resourcen, die die Application benötigt, und lädt sie gegebenenfalls. Die letzte Anweisung in der **!Run**-Datei lautet dann **Run <Obey$Dir>.!RunImage**, was das eigentliche Programm startet.

Die **!Run**-Datei von **!Paint** sieht so aus:

```
| > !Paint.!Run : Paint Version 1.09
     19-Dec-88
set Paint$Dir <Obey$Dir>
if "<System$Path>" = "" then Error 0
     System resources cannot be found
wimpslot -min 256K -max 256K
wimpslot -min 224K -max 224K
rmensure ColourTrans 0.00 rmload
     System:Modules.Colours
rmensure FPEmulator 0.00 rmload
     System:Modules.FPEmulator
rmensure sharedclibrary 0.00 rmload
     System:Modules.CLib
Set Alias$@RunType_FF9 run <Obey$Dir>
Set Alias$@PrintType_FF9 run <Obey$Dir>
     -print
IconSprites <Obey$Dir>.!Sprites
run <Paint$Dir>.!RunImage %*0
```



Zunächst setzt sich **!Paint** eine Kopie von **<Obey$Dir>**. Diese Variable enthält immer den Pfad zu der **Obey**-Datei, die gerade ausgeführt wird. Das bedeutet, daß ihr Inhalt nur vorübergehend den Pfad enthält, der **!Paint** interessiert. Deshalb legt **!Paint** eine dauerhafte Kopie davon in der Variable **Paint$Dir** an. Dieses sollte übrigens jede Application tun. Als nächstes folgt eine Kontrolle, ob die Systemresourcen bekannt sind, indem geprüft wird, ob es eine Variable **System$Path** gibt. Diese Variable wird von der **!System**-Application gesetzt. Jetzt wird nachgesehen, ob genug Speicher vorhanden ist, um das Programm zu starten. Dazu dient das Kommando **WimpSlot**. Es ist uns allerdings nicht ganz einsichtig, weswegen hier direkt hintereinander zwei WimpSlot-Kommandos stehen. In unseren Augen

wäre es sinnvoller, das zweite WimpSlot **hinter** die drei RMEnsure-Kommandos zu setzen. Dann wäre sichergestellt, daß zunächst genug Platz für Programm und Module ist, und nach dem Laden der Module wird dann nocheinmal geprüft, ob jetzt immer noch genug Platz für das eigentliche Programm ist. Die folgenden drei Zeilen laden Module, die das Programm benötigt. Auf Module will ich hier jedoch nicht weiter eingehen. Die beiden nächsten Zeilen sind interessant für Applications, die eigene Dateien bearbeiten (bei **!Paint** sind das natürlich Dateien vom Typ **Sprite**). Mit diesen Zeilen wird sichergestellt, daß beim Versuch, eine **Sprite**-Datei zu starten oder zu drucken, zunächst **!Paint** gestartet wird.

Neben diesen beiden Dateien gibt es in den meisten Fällen noch zwei weitere Dateien. Die erste ist **!Sprites**. Sie enthält das Sprite, welches im Directory für diese Application angezeigt werden soll, sowie wahlweise ein kleineres Sprite, das in den alternativen Directory-Anzeigen benutzt wird. Die Sprites heißen **!paint** und **sm!paint**, wobei paint in anderen Applications natürlich durch den Namen der Application ersetzt wird. Falls im Application-Directory keine !Boot-Datei (s.u.) steht, so wird **!Sprites** geladen, sobald das Application-Directory zum ersten Mal vom Filer "gesehen" wird (also das Directory geöffnet wird, in dem die Application ist).

Ist jedoch eine **!Boot**-Datei vorhanden, geschieht dieses nicht. Stattdessen wird **!Boot** ausgeführt. **!Boot** ist wie **!Run** meistens eine Datei vom Typ **Obey**. Eine Aufgabe, die **!Boot** erledigen muß, ist die Datei **!Sprites** mittels des Kommandos **IconSprites <Obey$Dir>.!Sprites** zu laden. **!Boot** von **!Paint** sieht folgendermaßen aus:

```
Set Alias$@RunType_FF9 run <Obey$Dir>
Set Alias$@PrintType_FF9 run <Obey$Dir>
     -print
IconSprites <Obey$Dir>.!Sprites
```

Wie man sieht, traten diese Zeilen bereits in **!Run** auf (eigentlich müßte es anderherum ausgedrückt werden: Die entsprechenden Zeilen in **!Run** traten bereits in **!Boot** auf. Dieses ist notwendig, da eine Application auch gestartet werden kann, ohne daß der Filer die **!Boot**-Datei "gesehen" hat, z.B. wenn die Application aus einer anderen **Obey**-Datei heraus gestartet wurde. Das typische Beispiel hierfür ist die **!Boot**-Datei des Systems, die nach dem Starten des Rechners ausgeführt wird. Viele Anwender starten hier bereits die Programme, die sie häufig benutzen. Wer die Abbildung betrachtet, wird noch zwei weitere Dateien im Directory von **!Paint** bemerken, **Sprites** und **Templates**. Auch diese Namen sind ziemlich standardisiert. In **Sprites** sind die Sprites enthalten, die das Programm selber benutzt (im Gegensatz zu denen in **!Sprites**, die vom Filer benutzt werden). Üblicherweise werden sie vom Programm selber kurz nach dem Programmstart geladen. Dieses gilt auch für die **Templates**, die Definitionen von Fenstern (Windows) enthalten, die das Programm benutzt. Aber das geht für diesen Artikel zu weit. Und deshalb ist er jetzt zu Ende.

Wenn eine Application etwas auf sich hält, so findet man eine Datei namens **!Help** in ihrem Directory. Wie der Name schon suggeriert, sollte `!Help` Hilfe bieten - unter RISC OS3 ist das besonders interessant, da der Filer bei Applications, die so Hilfe anbieten diese im Menü neben Copy, Rename, Count usw. anbietet.

(OK)

Ein paar BASICs

Hello World!

Der Klassiker aller Programme ist sicherlich das Programm, welches den Text "Hello World!" ausgibt. Dies ist in vielen Büchern über Programmierung das erste, was der Leser kennenlernt. Unter RISC OS geht es mit dem schönen BASIC recht einfach:

```
10 PRINT "Hello World!"
```

Allerdings hat das Beispiel eine Unschönheit: Acorns BASIC kann sehr gut OHNE Zeilennummern arbeiten. Dank Funktionen (FN), Prozeduren (PROC) und CASE geht die Programmierung problemlos ohne GOTOs und dann auch ohne die nach Spaghetti-Code aussehenden Zeilennummern, also:

```
PRINT "Hello World!"
```

FormFeed

Gelegentlich möchte ich meinen Drucker zu einem Seitenvorschub überreden. Klar, es geht auch, indem man am Drucker selbst die eine oder andere Taste drückt, aber per Maus-Klick wäre praktischer. Also wird BASIC bemüht. Das Programm unterscheidet sich vom Hello-World!-Programm fast gar nicht:

```
VDU 2,7,1,12,3
```

Dieser eine Befehl schickt dem Drucker einen Seitenvorschub. Warum das funktioniert, steht im BASIC-Handbuch - ich weiß nur, daß es so funktioniert.

Task oder nicht ?

Leider haben die o.g. Programme einen Nachteil: werden sie unter RISC OS gestartet (z.B. durch Anklicken), erscheint ein BASIC-Fenster, in dem das Programm läuft - und dieses Fenster verschwindet leider nicht automatisch, sondern erst, wenn Du die Leertaste gedrückt oder mit der Maus geklickt hast.

Es blieb mir also nichts anderes übrig, als das Seitenvorschub-Programm zu einer Wimp-Task, d.h. zu einem 'echten' RISC OS-Multitasking-Programm mit eigenem Fenster zu machen. Das war erfreulicherweise nicht allzu schwierig, da ich keine Bildschirmausgabe benötige, wenn ich dem Drucker nur einen Seitenvorschub schicken will. Folglich brauchte ich kein Fenster zu definieren usw. (Obige BASIC-Programme werden übrigens automatisch ähnlich gestartet, nur halt mit so einem unschönen Fenster und dem 'Hit SPACE ...' am Ende)

Das Prograämmle:

```
DIM Task%  5 : $Task% = "TASK"
DIM Block% 1024
SYS "Wimp_Initialise",200,!Task%,"FF"
    TO ,TaskHandle%

VDU 2,7,1,12,3

SYS "Wimp_CloseDown",TaskHandle%,!Task%
SYS "OS_Exit"
END
```

Der Aufruf von Wimp_Initialise startet eine Wimp-Task, das BASIC-Fenster kommt nun nicht. Mit Wimp_CloseDown wird meine Wimp-Task beendet und mit OS_Exit das Programm an sich beendet.

Absturz ...

Bei meinen ersten Gehversuchen mit BASIC habe ich meinen Rechner schnell zum Hängen gebracht. Das schafft schon folgendes Programm:

```
ON ERROR PROCfehler
DIM Task%  5 : $Task% = "TASK"
DIM Block% 1024
SYS "Wimp_Initialise",200,!Task%,"FF"
    TO ,TaskHandle%
A% = 8 DIV 0 : REM <--- siehe Text

DEF PROCFehler
  Block%!0 = ERR
  $(Block%+4) = REPORT$ + " at line" +
    STR$(ERL) + CHR$(0)
  SYS "Wimp_ReportError",Block%,1,"FF"
  SYS "Wimp_CloseDown",TaskHandle%,
    !Task%
  SYS "OS_Exit"
  END
ENDPROC
```

Warum? Die Zeile A% = 8 DIV 0 erzeigt klarerweise eine Fehler. Durch das ON ERROR oben wird in diesem Fall die Porcedure PROCfehler aufgerufen, die die Fehlermeldung via Wimp_ReportError ausgibt und dann das Programm beendet. Nur kommt es nicht so weit: PROCfehler gibt's gar nicht, sondern nur PROCFehler (mit großem F!). Das Fehlen von PROCfehler liefert einen neuen Fehler, also wird dank des OR ERROR-Befehls PROCfehler aufgerufen, usw.: das Programm loopt, der Rechner hängt!

(HzN)