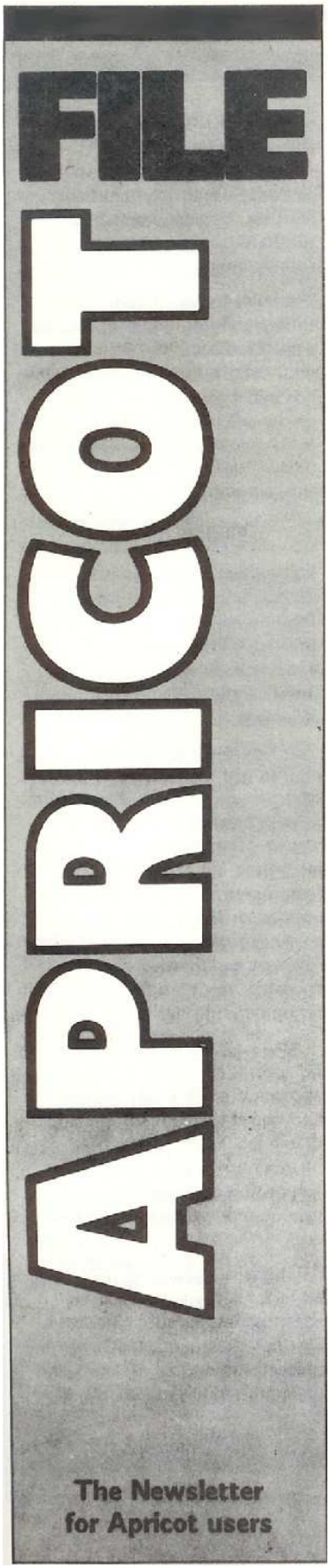
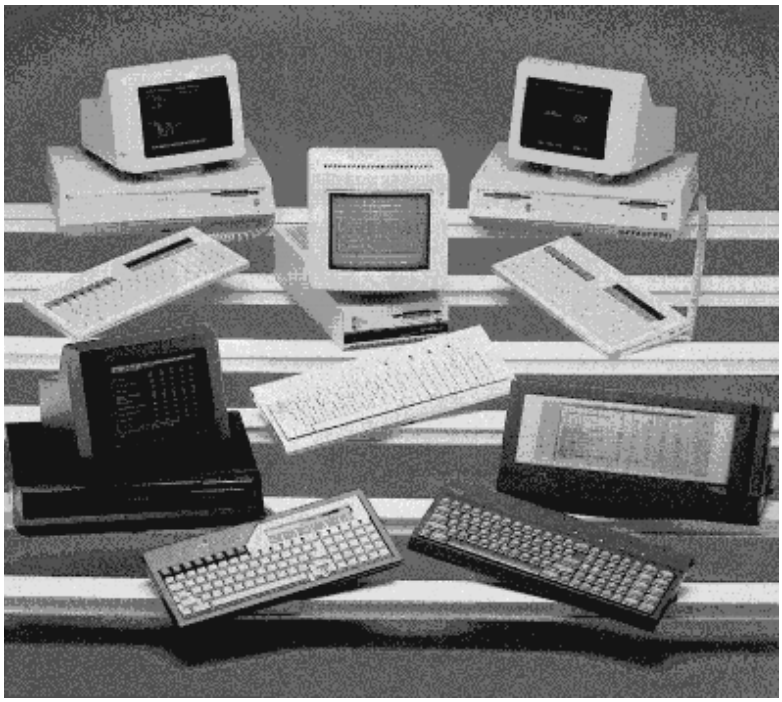


Apricot File 3.2



Scans courtesy of www.actapricot.org

Copyrighted content republished with thanks and permission from:

> Dennis Jarrett of TP Group
> Stuart Bunt
> Eddie Bromhead
> Mark Whitehorn

Many additional thanks to Eddie Bromhead for scanning all of Release 3, 4 and extra inserts.

Release 3.2

FILE APRICOT

The Newsletter for Apricot users

November 1987

Health warning:

Apricots can cause headaches

By Terence Green

I have lately taken delivery of my Apricot Xi HD, after a warranty repair at Apricot Computer Maintenance that saw a two-month spread between delivery and return. So I have a sorry tale to tell. You wanna hear it? Ok. But remember, *you* asked.

In late July my Xi started to act strangely during a reset — the machine seemed to have difficulty powering up. Switching on and off a few times eventually produced a working system; but things went from bad to worse quite rapidly — and within two days the Apricot had begun to reset automatically, apparently at random. The timing of this fault's appearance seemed providential as my warranty only had two months left to run.

Relieved that I still had warranty cover, I contacted my dealer who said it sounded like a fault in the power supply unit (PSU). He told me that he would arrange for the system box to be delivered to Apricot Computer Maintenance in Oldbury, West Midlands; and Apricot Computer Maintenance reckoned on a two-week turnround. Allow a week travelling there and back; that's three weeks in all.

Parting of the ways

On 23 July my computer was duly transported to Apricot Computer Maintenance by Securicor Parcel Express. August seemed to me a fine time to take a short break, so I arranged to rent a replacement for just two of the weeks I expected the computer to be away.

Three weeks to the day, on 13 August, my dealer informed me that Apricot had an item for him ready for collection, and he assumed it was my machine. He told me to expect it to be delivered the following Monday.

It wasn't; and when he checked its whereabouts with Apricot Computer Maintenance he was asked for proof of delivery because *they* had no record of my system unit.

On 21 August I wrote to Apricot to enquire after my computer. My dealer seemed by this time to have got himself grid-locked between Securicor who said they'd delivered and Apricot Computer Maintenance who said that they hadn't received it.

At this stage I had to drop everything for 10 days, part of which was spent in hospital (no, no, *not* for a surfeit of Apricots!). Everything came to a grinding halt.

On 1 September I took up the matter again with my dealer via a formal letter. Since the demise of the real Apricot range, he's moved on to other things, you see — training and networks — and he didn't seem to share my concern. So I asked him to confirm in writing the steps he was taking to return my computer and suggested some compensation for my expenses might be forthcoming.

CONTINUED ON PAGE 2

In this issue

About the Newsletter 3
The Disinformation Column: On WP Refinements 4
Batch File Recipes: Easier EDLIN 8
Turbo Pascal Primer — Part Two: Structured Programming . 10
Rebuilding the dream — re-riposte 14
Free Small Ads 16
The Workshop: Using the Apricot's sound generator 17
Writers in this issue 23
The Workshop: Changing from GSX to GEM 24
News of the Apricot world 28

Health warning

CONTINUED FROM PAGE 1

In fact I wasn't concerned about the money, I just wanted the computer back; but I thought this gentle pressure might be the right kind of lever to focus his attention on my problem.

And I also resolved to devote a whole day to tracing my computer. I made a good start. A series of calls to my dealer and Securicor Parcels Express confirmed that it had indeed been delivered. So, in combative mood, I called Apricot Computer Maintenance. They had no record of my computer. Sorry, they had no record. Sorry they had no record. I hung up.

Then out of the blue I had a call from a Mr Stan Cruft. He turned out to be the Goods Controller at Apricot Computer Maintenance. He was calling about my letter of 21 August. He was puzzled, he said. "Join the club", I thought. It seems my computer had been in the Despatch department since 14 August and despite a call to my dealer the thing still had not been collected! He said he would notify the dealer again, and he mentioned that whoever came to pick it up should ask for Mr David Venables.

Shortly after this call I was indeed told by my dealer that my machine would be picked up the next day, 3 September; and I would have it the day following. Brilliant!

Securicor went to Apricot Computer Maintenance on 3 September and asked for Mr Venables. He told Securicor that Apricot had no record of my computer.

Securicor tried again on 8th September, but again were turned away empty-handed. Apricot had no record ...

Back to the phone

On 14 September I called Apricot Computer Maintenance and was eventually put through to Mr Carl Millward, who said he was deputising while Mr Cruft was on leave. He traced my computer while I waited on the phone and he said "just give Securicor my name as the contact". I'd heard *that* one before, of course. But anyway I called Securicor; and they promised to collect the next day.

On 15 September, happy now in the knowledge that the Securicor van was once again trundling its way to Oldbury, I wrote a letter. I thought I would record my general feelings of dissatisfaction for the benefit of Mr Ian Clark, Customer Services Manager at Apricot. While I felt that it was individual Apricot staff who had been at fault in this whole sorry story, I nevertheless see it as the duty of management to be responsible for the satisfactory operation of the business. They get the credit when things go well, and there can be no 'right to manage' without concomitant responsibility.

So, I got it all off my chest, as they say — the needless delay, the phone calls, the inconvenience, the expense, my suffering work, the sheer ruddy frustration, blah, blah — and do you know, I felt much better? I eagerly awaited the morrow and the arrival of my Apricot.

Dong! Wrong again

All right, it's all very well for you to sit there and know me for a fool. You have the perfect vision that hindsight always brings. Personally, I like to think I'm an optimist, and I always have counted chickens.

Needless to say, my chicken didn't turn up.

So, on Saturday 19 September, I again put pen to paper. "Dear Mr Customer Services Manager, thank you for your nice, prompt, apologetic letter (good PR, that). But, Mr Clark, I have some bad news."

In this letter I demanded the return of my computer forthwith by the most direct means available, on the grounds that Apricot Computer Maintenance had a legal duty to return the machine within a reasonable time once the repair was done.

And for good measure I telephoned Mr Clark on the Monday (Freefone Apricot). He promised to look into it, and indeed called back with the details of the despatch note, the name of the Securicor driver who'd collected it, and the branch he came from. As it had obviously been picked up, he was surprised to hear I was still waiting. He promised to go down to Apricot Computer Maintenance and see what was going on.

He called back the next day to say that Securicor had delivered it to London W14 instead of N14. This it seemed was the source of all the problems, and I took delivery of my errant machine on 23 September.

When I saw the Apricot Computer Maintenance Service Report, the reason for the final delay was indeed obvious — my dealer's address was incorrectly entered as W14 instead of N14. But this was only the reason for the *final* delay; the rest of this farce was, and remains, something of a mystery. Incidentally, the engineer had signed the repair as being completed on 8 August!

To boot, I got my Apricot back in a very battered old box instead of the lovingly-preserved original packing I had used. If during various searches anybody had gone looking for my system's serial number, the tatty old box containing *my* machine but somebody *else's* serial number won't have thrown much light on the situation. Will it?

They all lived warily ever after

How can such a catalogue of errors be avoided? The temptation to say 'avoid Apricot plc like the plague' is powerful. But like many owners, I bought Apricot because it is undeniably a good design — not perfect, but a capable workhorse — and because it's British.

"Because it's British?" I hear you ask. Why? To keep the balance of payments in line? Well no, not exactly. With British Telecom and British Gas *et al* purchasing Taiwanese micros in corporate quantities, this approach is plainly whistling in the wind.

No. I bought British because I thought, in my wisdom, that I was preparing for the rainy days, for the times when things went wrong. Who better to turn to then a local manufacturer?

Perhaps my experience is unique? Conversations with other owners sadly don't support this theory. I met an Apricot user who instructed solicitors to effect the return of his Apricot after *four months* of deadlock!

But can general lessons be drawn from this? Well, the first year of Roger Foster's promise of five years support for real Apricots has passed and, frankly, I think that real Apricot owners are increasingly going

to have to put up with indifference to problems and requests, and second place in the service queue. I do hope you avoid the problems I've had. Use the checklist to minimise difficulties.

- **Make regular backups.** You've heard that before, but do you do it? rest assured that one day *will* need that backup, and urgently.
- **Make sure your maintenance agreement fully covers your needs.** Don't forget that breakdowns don't *ever* happen at convenient moments. Do you have access to another machine? You don't actually have a maintenance agreement? - whoops!
- **Arrange insurance to cover not only repairs but also the cost of renting a replacement machine.** This type of cover is increasingly available at reasonable rates; and it complements (rather than substitutes for) a maintenance contract. Rental firms with real Apricots are increasingly hard to find. It might even be worth buying a cheap IBM clone to use while your Apricot is being repaired. This isn't as daft as it sounds if you use software that was adapted from IBM software.
- **Obtain proof of despatch,** whether to your dealer or direct to Apricot.
- **Make sure you know when to expect your micro's return,** so you can start chasing immediately.
- **Don't believe that anything is settled until your computer is in your hands again.**
- **Always write to confirm telephone conversations;** this serves to establish dates too.

Apricot's service contracts include an 'exclusion of liability' clause. To my layman's understanding, this excludes *any* liability other than for death or personal injury arising from their negligence. Nevertheless, you have a right to demand the return of your computer equipment within a reasonable time of the repair being completed.

Afterword

After functioning perfectly for ten months until its visit to Apricot, my hard disk has begun breeding Bad Blocks at a remarkable rate since its return.

Good Luck. ◆

About the newsletter

Our aim is to provide a complete editorial and information service to users of Apricot computers.

Except in the normal course of day-to-day contact, sub sales and the like, we are not affiliated with or supported by Apricot plc or any of its dealers. Nor for that matter are we sponsored by or directly associated with any user group or any other such organisation, though our goals are similar to the declared aims of many such bodies.

The Publisher does not necessarily agree with the views expressed by contributors, and assumes no responsibility for errors of reproduction or interpretation in the subject matter of this newsletter or from any results arising therefrom.

APRICOT FILE is a publication of the TP Group — which until we get the name changed is the trading style of Permapoint Limited, registered in England no 1929195.

Postal address

TP GROUP
PO Box 509
London N1 1YL, England

Personal calls and deliveries

TP GROUP
Unit 4, 47-51 Wharfdale Road
London N1 9SE, England

Phone, E-mail, Telex

Telephone	**01-833 3501**
Telecom Gold	**81:JET767**
	[from overseas 10081:JET767]
Easylink	**19011170**
The Source	**BDE566**
Telex	**265871 MONREF G**
	quote reference 81:JET767

The People

Editor	**Dennis Jarrett**
Production	**Fiona Carey**
Writer	**Garret Keogh**
Subscriptions	**Terrance Duggan**
DFR/Technical	**Roger Gann**

The Guarantee

The annual subscription to APRICOT FILE buys 12 monthly issues as follows:

Britain	£50.00
Rest of Europe	£60.00
Elsewhere (air mail)	£65.00

If having subscribed you come to feel that APRICOT FILE does not meet your requirements or that it does not match our claims, we will gladly cancel your commitment to the magazine and refund the unused portion of your subscription.

■■■ ISSN 0268-6066 ■■■

The Disinformation Column

On SuperWriter and WordPerfect refinements

More advanced SuperWriting

By David Langford

Before the Ansible Information software list became too crowded for the trick to work, I used a simple SuperWriter variable-text document to print out our VAT invoices. The initial idea was straightforward enough: the overall layout of the invoice would be in one file imaginatively called INVOICE, and telling SuperWriter to print this would prompt you for more specific information, using the documented variables and embedded format commands. Something like this:

```
\GET NAME="Enter customer name:"
\GET PRODUCT="Enter product name:"
\GET PRICE="Price:"
\GET VAT="VAT content:"
Ansible Information (followed by the address, etc.)
Date: \INSERT A:TODAY
INVOICE TO \:NAME\
Product: \:PRODUCT\
Cost: \:PRICE\ received with thanks.
(Includes VAT at 15% -- VAT content is \:VAT\)
```

NAME, PRODUCT, PRICE and VAT are simple SuperWriter text variables. In theory one should be able to make PRICE a 'dollar variable', $PRICE, and use the supposedly built-in arithmetic functions to calculate a corresponding value for $VAT; but bugs in SuperWriter prevent this from actually working.

The \INSERT A:TODAY relies on our own little TODAY.COM, which writes a nicely formatted date into a text file called TODAY when you boot up; the equivalent system variable \%DATE unfortunately prints a numerical date in illogical American format, month/day/year. (It is possible to patch SuperWriter with DEBUG to correct this, as described by me here in the far-off mists of APRICOT FILE Release 1.7.)

So far this is all straightforward. It's worth adding a conditional test after the last of the GET commands and before any actual printed output, to allow an 'abort' if you've mistyped the input so far:

```
\IF VAT="x", PRINT OFF, ENDPASS
```

With this line in place, entering 'x' as the VAT value will cancel the coming print-out.

You can allow a simpler 'abort' by using a SET command immediately before the GET which prompts for the VAT amount: putting \SET VAT="x" would give VAT the default value of 'x', and hitting Return alone at the VAT input prompt would cancel printing by leaving this 'x' unchanged.

You might think that a test for a null string would be even simpler:

```
\IF VAT="", PRINT OFF, ENDPASS
```

This saves the need for the SET command just suggested. This is all right for the first invoice of a session; but if you print another without leaving the Print Menu, the VAT variable will have kept its value from the previous invoice. Which gives us a trick for doing successive invoice numbers, by the way: you can put in a \GET #INVNOnd and enter a number for the first invoice of the session; then, at the *end* of the main INVOICE file, put \SET #INVNO = #INVNO + 1 to increment the number automatically. This way, for the rest of the session, you just hit Return at the invoice number prompt.

The SHOW command can be used to remind you of the current default invoice number on a menu such as the one I'm shortly about to describe: \SHOW #INVNO will do the trick. Remember that all these variable names are arbitrary: #INVNO is merely a name I've made up for this example, and if you'd be happier calling it #SMEGMA then feel free.

Improving on the basics

Although you can go on endlessly adding small refinements to this variable-text file, they tend to hide the fact that the basic technique I've outlined is a trifle cumbersome. For example, you need to enter three chunks of text for each 'product' ... four, really, since this is a slimmed-down example showing just price, the VAT amount, and the total including VAT.

My next effort to improve things involved numbering the various possible products, and setting up a menu which would overwrite the Print Menu in SuperWriter, to offer various choices (here imaginary):

```
\CLS
\NOTE 1: Manic Space Goat Attack games software
\NOTE 2: Wire Brush for cleaning enemies' disks
\NOTE 3: Bombast Eradicator for use on other
contributors' prose
\GET #PRCODE="Enter product code (1-3):"
[Invoice header boilerplate, date etc. here]
\IF #PRCODE=1, INSERT GOAT.DAT\IF #PRCODE=2,
INSERT BRUSH.DAT
\IF #PRCODE=2 ...
```

And so on. The CLS (clear screen) and NOTE (display message) commands give a neat menu; the

IF conditions ensure that only the single chosen 'product' file will be inserted for printing, and of course the inserted file can contain the complete description and pricing details for the relevant product, at voluminous length.

A good way to simplify the contents of INSERTed files is to have them merely set variables for things like prices: if the insert files contain commands like \SET PRICE="45.00", the main document can use 'colon' output commands (like \:PRICE\) to print the variables — so that invoice layout is controlled by the main document, and to change it, you don't need to change all those INSERT files.

When using IF and INSERT, remember that several embedded commands (separated by commas) can be added after an IF — so that all will be executed if and *only* if the IF condition is met; while everything following an INSERT will be taken as part of the inserted filename, so an INSERT command should always be the *end* of a multiple embedded format line.

Ever so many more refinements are possible: there could be codes for combinations of two or more products, for example. The method is a little inelegant when there are many products: the block of IF tests becomes ever more ugly, sluggish and cumbersome.

What the manual didn't say

It was at this point that I discovered something you won't find in the manual: you can use a SuperWriter variable as the 'filename' in an INSERT command.

INSERT PRCODE won't work, whether or not PRCODE has been set up as a variable, since SuperWriter naturally takes this as an instruction to include a disk file called PRCODE. But INSERT #PRCODE is fine: the numerical *value* of the #PRCODE variable is treated as a file name. The entire block of IF tests can be condensed to \INSERT #PRCODE, provided that you name the example file GOAT.DAT as '1' (an unadorned number is quite legitimate as an MS-DOS filename), BRUSH.DAT as '2', and so on.

Then, wonder of wonders, I found by accident that 'hash' variables like #PRODUCT can be also used for text when doing things in this undocumented way. Instead of featureless numbers, the typed-in product codes can be memorable filenames — provided your embedded commands still pretend they're numbers by preceding the variable name with a hash sign:

```
[Menu commands here as before]
\GET #PRCODE="Enter product code (1 to abort):"
\IF #PRCODE=1, PRINT OFF, ENDPASS
[Boilerplate as before, including customer name output
and all that]
\INSERT #PRCODE
[More boilerplate, VAT number, etc.]
```

Although #PRCODE is officially a numerical variable, any filename typed in response to the GET prompt will be properly inserted. When #PRCODE is loaded with text, a numerical IF test will register its value as zero: hence the '1 to abort' option. (Of course, in this example I'm assuming that there isn't a product file named 1.) The Manic Space Goat Attack product information would now be in a file memorably called GOAT, MSGA, or maybe just plain G if you're a lazy typist.

You can go on elaborating like this forever. My next step was to allow an 'export flag' to be set as the result of another GET enquiry, providing for the fact that there's no VAT on export orders. For example:

```
\IF EXPORT="y",SET VAT="0.00",SET GROSS=NET
```

Note the use of the small 'y' to save your having to use the Shift key answering a \GET EXPORT prompt. If like APRICOT FILE you keep the total cost the same for export orders, to allow for airmail postage, the last part of the line becomes SET NET=GROSS.

I abandoned this whole system when the sheer number of possible product combinations became too vast. In theory it should have been feasible to allow the insertion of more than one 'product' file, to maintain cumulative totals of net prices and VAT, and to print out these grand totals at the end. Unfortunately the 'dollar' variables which allow decimal points (as indicated above) don't actually work. SET $TOTVAT = $TOTVAT + $VAT should do the trick but doesn't. SET #TOTVAT = #TOTVAT + #VAT *does* work, but requires that you either do your invoices in pence or arrange all prices to be whole numbers of pounds — neither being very practicable.

There are, I suppose, tortuous ways around this problem: you might keep pounds and pence in separate variables like #VAT and #VATP, and go through a regular series of 'carry' tests at each subtotal to prevent the pence from going into three figures:

```
\IF #VATP>100, SET #VAT=#VAT+1, SET
#VATP=#VATP-100
```

Formatting the final totals would also be a pain. Showing myself to be no true hacker, I gave up and wrote an all-purpose invoice program in Turbo Pascal instead ... but you might find some of the above techniques handy in SuperWriter form-letters and mailmerge.

Back at the ranch

Meanwhile, a user set me an interesting problem. He wanted an Execute file which would allow him to type (or load from disk) a list of filenames, and go away while SuperWriter automatically assembled

the named files into a single document. This was a tough one, made possible by the fact that the files in question were simple address records with no fancy formatting and which all ended in Return.

The eventual XQT file started from the edit screen (this was where you typed or loaded the list of filenames); used search-and-replace plus extra tinkering to convert the list into a series of \INSERT FILENAME embedded commands; saved this with a temporary filename; went to the Print Menu and selected Secondary Print Functions; formatted the document to disk so that the \INSERT commands automatically concatenated the files; left Print and reloaded the resulting file into the editor; cleaned out (with the aid of previously planted location markers) unwanted printer initialisation codes ...

If anyone thinks this is worth imitating, it's all yours. What I haven't yet figured out is why anyone should want to do such a bizarre thing in the first place.

Keyboard oddments

Why (they ask me) can't one program the Apricot's soft keyboard so that a favourite Execute file can be activated at a single keystroke?

It sounds simple: you only need four characters, ESC X a [Return], where the 'a' could be any legitimate one-letter filename: A, B, C and so on to invoke A.XQT, B.XQT, C.XQT ... The one-letter part is sheer economy, of course, not a SuperWriter or MS-DOS restriction: favourite .XQT files should always be given short names while esoteric ones get longer mnemonic filenames.

The trouble is that SuperWriter has a peculiar built-in restriction. I can't imagine why the program was written this way; maybe it's a legacy of some quirk in the CP/M operating system where SuperWriter's predecessors once gambolled.

The problem is that when ESC is read from the keyboard, there's a tiny built-in pause during which other keystrokes are not permitted. To put it another way, SuperWriter rebels (usually with a loud beep) if it finds more characters in the keyboard buffer after receiving ESC. Normally it's quite hard to type enough keys individually to run into this bug; but a modified keyboard puts in multiple characters more or less instantaneously.

(Remind me to insert a Slow Input switch in my ACOLYTE program, so that multiple keystrokes can optionally be inserted into the keyboard buffer in such a way as never to leave more than one character outstanding. I know this *can* be done, because I did recently write a 'screen capture' program which could feed an entire saved screen to the keyboard slowly enough to placate SuperWriter ... but it was a messy business, juggling between the keyboard and 'timer tick' interrupts to do the dirty work.)

Useful escapes

So programming a key to generate ESC plus something else is bad news in SuperWriter. It can be useful in MS-DOS, though; I keep running into people who haven't unraveled the obscure table of 'special command line editing keys' in the depths of the manual. Next time you modify your basic MS-DOS keyboard, try assigning ESC U to the REPEAT key (it must be a capital U).

In Apricot MS-DOS this restores the last command line you typed in, which is useful if the line was complicated and correct but failed with a 'bad command or file name' message — swap to the proper disk and hit REPEAT followed by Return. You can also backspace after ESC U to edit a previous command slightly: it's worth experimenting. (Try making Shift-REPEAT generate ESC U *and* Return, for instant replay.)

There are several fiddly ways to edit the command line, but in years of Apricot use I've rarely needed the more exotic ones. ESC T followed by any character has its uses, restoring the previous command up to (just before) the selected character; ESC P lets you insert characters rather than (as usual) overwriting the former command line.

By this stage, though, it's generally simpler to type the damned thing in again rather than remember these control codes. ESC U and its brethren remain most useful for amazing and impressing less experienced users who are looking over your shoulder.

WordPerfect corner

As I've remarked before, WordPerfect 4.2 is somewhat inconsistent in its handling of search-and-replace operations which involve print controls and format commands. I ran into a particularly odd example with the >Indent< command (keyed on Apricots with CONTROL plus the dot on the calculator pad). This works like a hanging indent but also indents (leftwards) the righthand margin of the paragraph.

One day, in a document full of indented quotations, I decided that Replace would be a good way to indent them all further, by swapping each single >Indent< code for a pair.

I approached this cautiously, knowing that WordPerfect won't accept every special code when Searching: but all seemed well, the magic code [->Indent<-] appearing at the 'Srch' prompt when I hit the indent key. Full of confidence, my fingers rushed on with ESC (to end the search text), two more indent commands (to replace the single one) and ESC again (to kick the search-and-replace into action). More moments of 'please wait' later, there was the usual smug beep — whereupon I found that all my indented paragraphs had come undone and slipped back to the normal margins.

It turns out that WordPerfect accepts the >Indent< code at the 'Srch' prompt, but *not* at the 'Replace with' prompt; so I'd replaced them all with neat parcels of nothing at all.

Another oddity — useful this time — is the handling of boldface and underline codes in search-and-replace. Hit the Bold or Underline keys, and the code [Bold] or [Undrline] dutifully appears after 'Srch'. Using these search codes will find the *beginnings*, only, of boldface or underlined blocks. But press Bold or Underline a second time at the same prompt, and this time what appears is the little [b] or [u] for the *end* of boldfacing or underlining. The idea is of course to let you specify a particular word or whatever: [Bold] and [b] appear alternately, whether or not text has been typed in between.

It was Chris Priest's idea to pervert this facility by entering a keystroke sequence such as ...

Search
Bold
Bold
Cursor Left
Delete Left (ie Backspace)

This enters and then wipes out the [Bold] code, leaving the little [b] as the sole 'character' to be searched for. In other words, you can search for the *ends* of boldface or underlined chunks of text... which may not be a mindbogglingly useful operation when working from the keyboard, but has some very handy applications in macros.

A tiny bit of news on the WordPerfect front: after long delays, we were finally able to get hold of Spanish and French spelling-check lexicons for a polyglot who wanted them (other languages are presumably available). These cost £60 apiece plus VAT, the package consisting of one (yes one) disk carrying the appropriate version of LEX.WP plus a SPELL.EXE program with foreign-language menus and prompts. The accompanying documentation covers 0 (zero) sheets of paper.

Legal fictions

Just for those who asked: Ansible Information did eventually survive the savage assaults of Sentinel Software and WordPerfect Corporation, who'd managed to get the impression that we were flogging illegally modified versions of their copyright software (actually we introduce corrections via independent 'background' programs) and spelling lexicons (actually we converted the literate dictionaries we'd developed for SuperWriter). And having accepted our unshakable integrity and cuddly niceness in these areas, the WordPerfect Corporation solicitor decided to earn his keep by forbidding us to use the word 'WordPerfect' ...

Well, it's slightly more complicated and unintelligible than that, and for the benefit of other entrepreneurs I'd better explain. *Their* position is that 'WordPerfect' is a vital trademark, a valuable asset of the company, which must not be taken in vain. Thus 'Ansible WordPerfect Disk' is deemed to be very wicked, and we've been required to change to some completely different name: 'Ansible Disk For WordPerfect' would apparently be ok (it being completely different).

After toying with legally unassailable but rather naughty possibilities — 'Ansible Word Perfect Disk', 'Ansible W*rdP*rf*ect Disk' and 'The Disk WordPerfect Corporation Paid An Expensive Lawyer To Prevent Ansible Information From Naming' — we inexplicably settled for the boring name PARAGON (subtitled 'Ansible WordPerfect Enhancements' or possibly, to be on the safe side, 'Ansible Enhancements For WordPerfect').

It was paranoia in the face of this legal intimidation which persuaded us to become a limited company: our accountant supplied an existing company off the peg. For a terrible two months before we got it changed, Ansible Information was saddled with a company name which sounded like a strip-o-gram delivery service.

Mind you, after following the æons-old history of a certain outfit which 'until we get the name changed is the trading style of Permapoint Limited', we were afraid it must take *at least* two years to alter a company name ...

(Oh all right, we'll get around to changing the company name. And there's more about PARAGON *and the Ansible vs WordPerfect struggle in Chris Priest's* Re-Riposte *piece elsewhere in this issue — Ed.)*

The famous PC-286 upgrade

I really am not keen on the idea. In perspective, the way it looks to me is this: we're being given an opportunity to trade in our old Apricot PC/Xi machines for a few hundred pounds' discount on an IBM AT clone — but not a *new* one. In many ways it's a second-hand AT, with your old grubby keyboard, monitor, systems box, and the rest. The Hercules monochrome video standard is virtually a dead duck: even Amstrad's cheapo IBM imitations are going over to the EGA standard now. And then there's all that expensive IBM software to buy.

If I needed the speed (not vital in word processing), I'd rather pay more for an AT clone and still have my Apricot as a reserve machine. If it's simply immediate IBM-compatibility that's required, the cost of the conversion job will buy a perfectly good XT clone with the maximum 640K memory and a hard disk — brand new. And you keep the Apricot.

When pressed, I chose the second option, with a Ness Turbo (speed switchable between 4.77 and 8MHz; nifty amber monitor; I/O card included just as though it were an Apricot; Hercules graphics which I don't use). To this, I've fitted a cheap add-on 3.5in drive for convenience in file transfer — Apricot BIOS 3.1 happily reads and writes IBM-format disks. I still love and prefer the old Xi, though. Honest. ◆

Batch File Recipes:

Easier EDLIN

By Roger Gann

Just as MS-DOS is closely modelled on the CP/M-86 operating system, so EDLIN bears a close resemblance to its CP/M forebear ED. As a result, it has an obscure command structure and is rather slow and awkward to use. It is therefore often relegated to minor editing tasks such as preparing batch files.

Even so, using it can be a bit of a pain. This is compounded by its poor implementation on the Apricot. Other manufacturers have made an effort; on the IBM PC, for example, the function keys F1 to F5 as well INS and DEL are used by EDLIN (and for editing the COMMAND line too) to perform specific editing tasks. The function key features were even carried over to the Sirius, the Apricot's big brother. But they never made it to the Apricot.

At a recent press bash I confronted Apricot's technical supremo Peter Horne with this point. Sadly the mists of time proved too dense for him to shed any light on this matter. Anyway, as far as can be made out, it seems that when the original BIOS was being written, the chaps at Advanced Tech sought assistance from MicroSoft in the UK as to how to implement the editing function keys. Difficult as it may be to believe, for once, MicroSoft UK knew less about it than Apricot; and the editing keys never came to be.

So Apricot owners were deprived of some really useful features. Determined EDLIN users had to learn obscure key combinations like ESC T and ESC U to make full use of its very limited repertoire. This sort of problem cries out for a solution involving two of the better features of the Apricot PC: its 'soft' keyboard and the MicroScreen.

Help from the Xen-i

The original Xen-i machines featured MicroScreens on their IBM-style keyboards and additionally came with MS.EXE, a utility that replaced the original MicroScreen configurator MISCREEN.COM. Not only is it much better than its predecessor, by golly, it runs on the PC/Xi to boot!

We've included it on the new DEALS FOR READERS disk AP027. MS gives you three options — you can define individual MicroScreen keys on the fly, you can use it create a 'script' file for MicroScreen assignments, and you can load a previously-written script file:

`MS keynum "text" "string[@]"`

keynum is the MicroScreen key number in the range 1 to 6 (reading from left to right); *text* (inside double quotes) is whatever you want to appear on the MicroScreen above that key; *string* (again in double quotes) will be assigned to that key — the optional @ character will be replaced by a Return.

`MS /E filespec`

the /E signifies the edit option for amending or creating a script file with the name given. A script file is nought but a common-or-garden ASCII file with the suffix .MSN; you could also create such a file using (what else?) EDLIN — or for that matter any word processor that can produces straight ASCII files.

`MS filespec`

this one simply loads and uses the MSN file nominated.

A special MicroScreen keyboard file called say EDLIN.MSN might look like this:

```
COPY1 COPYUP COPYALL SKIP1 SKIPUP  VOID
1
111111
0;144;27;"S"
0;148;27;"T"
0;152;27;"U"
0;156;27;"V"
0;160;27;"W"
0;164;27;"H"
```

The first line contains the actual MicroScreen labels — note that these will be confined to the bottom line of the display when they appear. The lone digit '1' determines whether or not you want the date and time displayed (a '0' will switch them off). The third line contains six 1s; these represent the red LEDs to be switched on (again '0' will leave them off).

The remaining six lines relate to what the keys will actually do when pressed — the 'key assignments'. The first two sets of digits (0;144) seem to be constant and probably are the ASCII value of the scan code that each soft key generates.

The rest of the line represents the command string that each key will generate. Each part of the command string is separated by a semicolon. The first key in our MSN file generates the string representing the 'COPY1' command permitting you to copy characters one at a time from the template to the new line. This string is ESC S; you would normally press the ESCAPE key and 'S' to achieve this.

Note that the 'S' and other QWERTY characters are enclosed in double quotes. Non-printing characters can also be used by quoting their ASCII value in decimal — thus Escape is represented by '27' in our example. Returns are '13', additionally followed by a lower case 'p' if it terminates the line. So to change to drive A and run a file called SW.BAT, the string would look like this:

```
0;144;"A:";13;"SW";13p
```

So the MicroScreen has a total of six EDLIN editing functions. I could have put INSERT (ESC-P) on there as well; but the keyboard already has a perfectly acceptable INS key, and it would be simple enough to assign this single key using ESC 4 with a line in EDIT.BAT to effect this (it may even be possible to fool MS with a spurious scan code value).

Alternatively, you could opt for a separate EDLIN.KB file, prepared using ACTIVITY, which reassigns this key as well as the MicroScreen keys. This may seem like gilding the lily, setting up the MicroScreen keys once with MS and again with KEYS; but it does give users the opportunity to dispense with the MicroScreen display if required.

Only one other EDLIN function remains to be implemented: ESC Q. This is the 'replace' function and merely turns off insert mode. As it is the default setting for EDLIN, I have assigned this string to SHIFT-INSERT on the editing keypad.

Using EDIT.BAT

To edit a particular file you simply type 'EDIT' followed by the filespec. So to edit AUTOEXEC.BAT you would enter ...

```
A> EDIT AUTOEXEC.BAT
```

The batch file EDIT.BAT looks like this:

EDIT.BAT

Line numbers are included for reference — don't enter them when keying in this batch file.

```
1   ECHO OFF
2   CLS
3   ECHO  Now installing MicroScreen display
4   MS EDLIN.MSN
5   CLS
6   ECHO  Now installing EDLIN keyboard
7   KEYS EDLIN SAVED
8   EDLIN %1
9   KEYS SAVED
10  DEL SAVED.KB
11  MS BLANK.MSN
```

The first two lines turn off the screen 'echo' and clear the display. Line 3 tells you what's going on and line 4 loads the MicroScreen display for EDLIN. The screen is cleared again, you're told that the EDLIN keyboard is being installed, and line 7 does that — with the existing layout saved as SAVED.KB. Line 8 loads EDLIN with the file to be edited, represented by the replaceable parameter %1.

When you leave EDLIN, line 9 reinstates the previous keyboard; there's no point in saving this file, so it goes (line 10) and line 11 loads an empty MicroScreen display that also reverts to its previous state.

Obviously for this batch file to work properly, you must have EDLIN, MS, the MSN script file and KEYS (version 2.0) on the default drive or accessible through the PATH.

On cat-skinning

You can overdo it, though — repeated use of MS in a session can result in a 'no room in keyboard table' error message: I guess this means you've exhausted the space available in RAM for keyboard tables.

Nice and convenient though it is, you don't actually *need* MS.EXE. If you use the special EDLIN.KB, all you need to do to kick the MicroScreen into life is label it.

Now, you can use all sorts of Escape codes to achieve this; but a far simpler way is to copy a text file to the MicroScreen. As 'Hank' Langford has pointed out in his *Disinformation Column*, the MicroScreen is an MS-DOS device — well, it is on Apricots at any rate. And just as you can copy a file to a printer, you can do the same with the MicroScreen.

The file to be copied — MSLABEL, say — looks like this:

```
1
2  COPY1 COPYUP COPYALL SKIP1 SKIPUP  VOID
```

(Line numbers are included for clarity only.) Astute readers will have noticed that line 1 is blank. This is because the MicroScreen display can accommodate two lines of 40 characters; putting the desired labels on line 1 merely results in them rapidly scrolling off the MicroScreen leaving it completely clear. The solution is to put the labels on the second line, which is where I wanted them anyway.

This is the revised batch file:

EDIT2.BAT

Line numbers are included for reference — don't enter them when keying in this batch file.

```
1   ECHO OFF
2   CLS
3   ECHO  Now installing MicroScreen display
4   COPY MSLABEL MSCREEN
5   ECHO  ^[/?
6   CLS
7   ECHO  Now installing EDLIN keyboard
8   KEYS EDLIN SAVED
9   EDLIN %1
10  KEYS SAVED
11  DEL SAVED.KB
12  ECHO  ^[/@
13  ECHO  ^[S
```

This is very similar to EDIT.BAT except that the fourth line sends the labels to the MicroScreen, the fifth line is an Escape sequence which turns on all the red LEDs, line 12 turns them all off again and the last line restores the date and time to the MicroScreen. Don't forget how to produce the Escape symbol (^[) using EDLIN: use ESC R (not lower case!). It's amazing what you can do when you try ... ◆

A Turbo Pascal Primer

Part Two:

All About Structured Programming

By Mark Whitehorn

While writing this series on Turbo Pascal, I found myself continually wishing to digress in order to explain why I was programming in a particular way. Eventually it seemed more reasonable to digress completely for a while, and talk about structured programming.

I started programming in the usual way — sitting at a terminal and throwing code at the computer until a program appeared. This was great fun and it was from this sort of work that I acquired my love of programming. Many people program in this way and regard people who tell them to program 'properly' as party-poopers. Quite right, too — you shouldn't take anyone's word about the 'right way' to program. I became a fan of structured programming only when it started to be to my advantage to program in a more structured way; so instead of resenting the more rigid discipline, I loved it because it made my life easier.

How do you know when it starts to be to your advantage to use structured programming? Easy — when you realise that you cannot understand your own programs. Does this look familiar?

```
Repeat
If X>Y and Flag1 then Procedure4; If F>X then
  Procedure2 else Y:=X;
If X>42 then F:=F+1; Procedure3
until F>60;
```

It's 2am and your brain hurts. The program was originally intended to be part of a file editor. You are working on a bit which is supposed to wrap the text round, and it doesn't. You have forgotten what X stands for and are sure that you have not used a variable called F, despite the evidence above. Y is the length of the string before wrap around, so why have you made it equal to X if F is not greater then X? Should the bottom of the loop be until Y>60?

You get the idea. Once your programs get to the stage where debugging them is becoming a real pain, a little structuring would help you to carry on enjoying the process.

Another pragmatic reason for adopting Structured Programming concerns employability. Several surveys suggest that maintenance accounts for 80 to 90 per cent of the total cost of a program. Structured programming produces programs which are much easier to maintain, and many firms are now insisting that their programmers adopt these techniques.

An overview

It is important to realise that programmers usually have two distinct perspectives of the program they're about to write. One is a high-level view; that is, you understand in broad general terms what the program is expected to do. The other is a low-level view — you have an understanding of the computer language you will be using. For example, you know what `Writeln('Penguin');` will do if it's inserted into a program.

So, one problem is that you have to bridge the gap between these two levels without getting completely lost. Structured programming helps there.

You must also appreciate that human beings are bad at coping with too much information at one time. When we program badly, we tend to try coping with the program as a whole. This is fine for small programs but impossible with large ones; structured programming helps to keep the program in manageable chunks at all times.

The tools

There are three vital tools to help you improve the structuring of your programs.

- **Procedures:** Languages such as Turbo Pascal allow you to isolate a piece of code and give it a name. You are then allowed to utilise that piece at any time within the program, simply by using its name. (That also applies to functions; throughout this article, read 'function and procedure' for 'procedure'.)
- **Names:** Most programming languages allow you the freedom to assign names to both procedures and variables.
- **Comments:** Again, most languages allow the programmer to insert comments into the code. These serve to remind you what's going on, and they provide invaluable clues for those who try to maintain the program in later months.

Like all tools, however, these can be misused. When a program is well-structured, these three are used together, in balance, so that each one complements the others.

Top-down programming

And finally, we come to Structured Programming Itself. 'Top-down programming' is the *heart* of structured programming.

The overall idea is to start off breaking the problem into smaller bits. Then take one of those bits and break it down into smaller bits. Continue this sub-division process until the bits are of manageable size. The expression 'top-down' refers to the fact that we start at the top level of understanding and work our way down to the lower level, which is the code itself.

To do this in practice, start off with a list of the tasks you wish the program to perform. As an example, consider a program to plot a series of points on a graph and draw a line of best fit through them. I want the program to do this lot:

1. Put up a title page.
2. Explain that the program is in the public domain.
3. Ask for the data to be fed in by the user.
4. Take in the data until the user has finished feeding in points.
5. Plot the data on the graph.
6. Calculate and draw the line of best fit.
7. Print the graph if the user wants a hard copy.
8. Repeat steps 3 to 7 until the user has plotted enough graphs.
9. Sign off in a friendly manner.

This is level 1 (or level 0 if you are a computer pseud). All of these steps will become a separate procedure in the finished program. I then give each of these procedures a sensible name, one which reflects its function. Having done this, I can write the main program.

```
Var
FinishedGraphs : Boolean;
Begin                        {Main Program}
TitlePage;
PublicDomain;
Repeat
    InputData;
    PlotData;
    LineOfBestFit;
    HardCopy;
Until FinishedGraphs;
SignOff
End.                         {of Main Program}
```

I think that this is reasonably clear and readable. A vital point is it realise that the procedural names are chosen so that they provide most of the documentation. Comments like "This program puts up a title page, and then a notice about the program being Public Domain" should be superfluous.

Also note that it will be easy to change this program in the future. If, for example, a user wants the facility to add points to a graph after plotting, it is relatively easy to add more procedures:

```
Var
FinishedAddingPoints,
FinishedGraphs : Boolean;
Begin                        {Main Program}
TitlePage;
PublicDomain;
Repeat
  InputData;
    Repeat
      PlotData;
      LineOfBestFit;
      AddPoints;
    Until FinishedAddingPoints;
  HardCopy;
Until FinishedGraphs;
SignOff
End.                         {of Main Program}
```

This is Level 1 completed: I have a clear idea of the overall way in which the program will solve the problem.

Now I look at one of the bits in more detail. Take for example the first procedure, TitlePage. This has been called from the main program (which is level 1) and hence is considered to be part of level 2. Any procedures named in TitlePage would be part of Level 3, and so on.

The process is repeated; the steps which the procedure is expected to perform are listed.

1. Clear the screen.
2. Position the text.
3. Write a greeting.

Now I have to decide whether these three steps can be sensibly performed within one procedure. If not, I simply call each step a new procedure, and then examine each of those procedures in turn; this would be going down to Level 3. As it turns out, these three steps can be performed within one procedure, so the process can stop at level 2. All that remains is to write this procedure in Turbo Pascal.

Of course this doesn't mean that all other procedures from level 1 must be broken down only as far as level 2. The procedure LineOfBestFit for example, could easily go down to level 4 or 5.

Bear in mind that Level 1 is not sacrosanct. If I decide, as the programming proceeds, that it would be better with a slightly different layout, then I can always change Level 1. The important point is that I should always have a simple, clear idea of how the program is laid out.

Procedures

An important consideration is how you decide *which* bits of the program should go into separate procedures. I tend to apply the following general criteria:

- A piece of program should be a bottom-level procedure (that is, one which does not call

procedures from a lower level) if it does a discrete job, is less than about 50 lines long, and is not indented too heavily.

- Some procedures are used frequently within the whole program, often being called from different areas of the program. These are usually better placed in a 'library' (see below).

Beyond this, you are on your own. This is where a lot of the skill in programming comes, and there is no substitute for experience. Do not be afraid to experiment, and be prepared to change your ideas during the development of the program.

Names

Variables, like procedures, should have sensible names. Wherever possible, try to make variables local (that is, having meaning only within a single procedure); global variables should be reserved for passing information between procedures. This reduces the apparent complexity of the program, so that it is more easily understood. Like many of the devices described here, this is really a trick — the program doesn't become simpler, it becomes more understandable to human brains. The computer can understand it either way, but we are trying to help humans.

Comments

Your program should be commented for the benefit of those who have to maintain it — this may well be you in six months' time!

There are differing opinions about the number of comments which should be placed in the code. But some general comments at the beginning of the program are always helpful; author, version number, date of last change, general overview of what it is supposed to do — anything that you think might be useful in the future.

I prefer to put in some lines in at the beginning of each procedure and try not to clutter the code with too many comments. As I said before, the procedural and variable names should provide much of the documentation. These are often enough since each procedure shouldn't be more than 50 lines long (often shorter). On the other hand, I don't hesitate to put comments into the code if I think it will be useful. I always imagine a complete stranger reading it; this seems to help. But as with the size of procedures, practice makes perfect.

Indenting

All loops within programs should be indented as shown in the Repeat/Until loops which are used above. Most people seem to do this anyway, but I mention it to point out that indenting is considerably aided by top-down programming.

One complaint about indenting is that the whole program waltzes across the page and ends up crushed against the right hand margin. By splitting up the program into procedures, the number of loops in any one place is reduced.

Procedural position

A program written in the way outlined here will produce a series of procedures, and it would be foolish to leave these scattered around like leaves in Autumn.

The requirements of Turbo Pascal give us a perfectly sensible order which we can adopt. To avoid the use of forward declarations, we must first declare those procedures which are called by other procedures. Thus the lowest level procedures (paradoxically the ones with the highest number) are declared first, and they should be labelled as such.

Libraries

As you continue to program, you'll probably produce a whole series of procedures which you use commonly in a variety of programs, and which are likely to be called from procedures on all levels. Here's an example of a procedure I find particularly useful:

```
{-------------------------------------------}
Procedure Press;
{This is a typical end-of-page sort of procedure}
Begin
  GotoXY(5,24);
  Write('Please press any key to continue.');
  Repeat until KeyPressed;
  ClrScr
End; {Of Procedure Press}
{-------------------------------------------}
```

These may be put into 'libraries' which stand outside the hierarchical system I have described. Such libraries are included at the beginning of the program and the procedures can be called from anywhere within the program.

I have placed a collection of these procedures in a file called USEFUL.PAS, and I can include them by placing the statement {$I USEFUL.PAS} just before the first procedure in the program.

Last words

Well, that is a quick résumé of structured programming. Do remember that these are only guidelines; be flexible — sometimes there will be another way which is more readable or seems more elegant. Try your own ideas, and have fun. ◆

Now read on ...

I feel guilty about not completing the program from last month, so here is a solution. I will discuss it in more detail in the next instalment, but I've stuck in some detailed comments which may help.

```
Var
  InputFile,OutputFile  :text;
{-----------------------------------}
Procedure TitlePage;
Begin;
  ClrScr;
  GotoXY(5,5);
  Writeln('Please enter the name of the source file.');
  GotoXY(5,12);
  Writeln('Note that you must give the complete file spec. e.g. B:Penguin.txt .');
  GotoXY(8,18);
End; {Of procedure TitlePage}
{-----------------------------------}

Procedure OpenFile;
{This ties the name 'InputFile' to the Input file chosen by the user, and ties The name 'OutputFile' to the
output file. In this case the output file is CON (the console) so the output appears on the screen. So
whenever the name InputFile is used in the program, it is really referring to the input file, and OutputFile
refers to the screen.}
Var
  Source                :String[30];
  Target                :String[30];
Begin;
  Readln(Source);
  Assign(InputFile,Source);
  Reset(InputFile);
  ClrScr;
  Target := 'Con';
  Assign(OutputFile,Target);
  Rewrite(OutputFile);
End;{of Proc OpenFile}
{-----------------------------------}

Procedure RemoveCharAndDumpToScreen;
{This Procedure removes characters from a file and examines them. If they are not writable (that is, have
ASCII codes 1-12,14-31, or 127-255) then a space is written to the screen. All writable characters are dealt
with by the Else statement, and are written to the screen. All of these characters are written on the same
line, which would be silly if it went on for too long. To cope with this, when an ASCII code 13 (carriage
return) is detected, a Writeln; is used, which causes the text which follows to appear on the next line.}
Var
  TempLet               :Char;
  Ascii                 :Byte;
Begin;
While Not EOF(InputFile) do begin;
   Read(InputFile,TempLet);  {Takes in a character from infile, the input file}
   Ascii := ord(Templet);    {Sets "Ascii" to ASCII code of the character}
   Case Ascii of
     1..12    : Write(OutputFile,' ');
     13       : Writeln(OutputFile);
     14..31   : Write(OutputFile,' ');
     127..255 : Write(OutputFile,' ');
     Else       Write(OutputFile,Templet);
   End;
End;
End;{of Proc RemoveCharAndDumpToScreen}
{-----------------------------------}

Procedure CloseFiles;
{Not really a tricky one. It closes the input and output files. The output file in this case (CON) doesn't
need to be closed; but I closed it here to remind you that if a real file is used for output, then it has to
be closed.}
Begin;
Close(InputFile);
Close(OutputFile)
End;{of Proc CloseFiles}
{-----------------------------------}

Begin;            {MainProgram}
{BE WARNED — THIS PROGRAM HAS NO ERROR CHECKING AS YET. USE ONLY ON COPIES OF FILES, NOT ORIGINALS. DON'T
BLAME ME IF IT MELTS YOUR HARD DISK. Note also that I have amended my original list of four procedures (as
shown in last month's issue). Of this, more next month.}
TitlePage;
OpenFile;
RemoveCharAndDumpToScreen;
CloseFiles
END.
```

Though this program needs much more in the way of error-checking, it *does* actually work ... though according to Borland's Turbo Tutor, it shouldn't. A prize of all the APRICOT FILE DEALS FOR READERS public-domain Turbo Pascal disks for the first reader to tell us why it *shouldn't* work.

Re-riposte

Rebuilding the Dream

A response to Brian Read

By Christopher Priest

In APRICOT FILE Release 3.1, Brian Read took my venerable partner Dave Langford to task on the respective merits of SuperWriter and WordPerfect.

I once coined a computer word, 'YGUTWYG'. This means You Get Used To What You've Got, and describes a rare computing condition: user contentment. The prosperity of the computer industry is actually based on creating envy and discontent amongst its customers to encourage reckless purchasing. As a result, anyone who is happy with what he's got is considered to be a bit of a maverick.

Although Dave Langford and I, as 'Ansible Information', are a part of this terrible industry, we retain a great regard for mavericks. In Mr Read's case, if he's content with SuperWriter there's no earthly reason why he should change. All software produces a subjective response, and in this sense Dave and I have no axe to grind for either SuperWriter or WordPerfect.

For instance, we have always maintained that SuperWriter is a good middle-range word processor — classically designed, underrated not only by Apricot who supplied it but (oddly enough) by Sorcim who wrote it. As a result it's rather neglected by those who have not tried it. We continue to produce our **Ansible SuperWriter Disk** and still use SuperWriter for selected tasks.

WordPerfect is a more advanced program, more 'powerful', crammed with more features ... but not everybody loves it. I'm one of those who do, as is Dave Langford.

But Mr Read's article needs a response, because (for one thing) he seems fairly insensitive to irony.

Background

Dave's article was indeed partly about the "coruscating wonderfulness" of WordPerfect (against which SuperWriter does frequently seem a bit uncoruscating). But surely Mr Read realises that this was laid down as the background against which a few foreground reservations could be placed in context? Those reservations arose because of WordPerfect's interesting relationship with the 'generic', pre-IBM-compatible Apricot.

As two people who have taken a great interest in the Apricot market, we've been getting pretty sore. Our argument goes like this:

- A lot of Apricots were sold in their heyday.
- Nearly all of them are still in use.
- Technically, Apricots are still holding their own against the competition of IBM clones.
- Most of the people still using Apricots appear to want to go on doing so, in spite of discouragement from everyone in the world (including Apricot plc).
- These people are now very badly served by large software houses. (For example, Microsoft Word 3 is not available in Apricot format and never will be.)

What is unusual about WordPerfect is that it *is* available in generic Apricot format. WordPerfect Corp has cleverly cornered a market: WordPerfect is the most advanced word processor available for Apricots. A shrewd move. No argument.

Where the argument starts is with the UK distributors of WordPerfect. Not only do they appear to know nothing about the special demands of the Apricot (the variant BIOSes, for example); they proudly and loudly declare they have no interest in finding out what they are — and, moreover, they appear to wish that this huge army of users would quietly go away.

We at Ansible Information like telling the story about our outburst of innocence a year or so ago, when we asked one of the distributors' technical ladies if plans existed to release WordPerfect 4.2 in Apricot format. (At that time it was by no means certain.) Everyone in the room turned our way in embarrassed silence. Then the young lady said, "Oh nobody uses Apricots these days! Tut-tut!" ... and everyone sniggered with relief.

Bugs and blindness

WordPerfect 4.2 *was* eventually released in Apricot format. Mind you, it wouldn't display anything under BIOS 2.4; and screen output was unreliable on the F2 and F10; and the SPELL program didn't run properly; and the Thesaurus wasn't supplied; and anyway even if they could obtain it somewhere people with single-sided disk drives couldn't cram the Thesaurus on to a disk ...

Not only were the UK distributors incapable of fixing these problems, *they didn't even know they existed.*

This, then, is what Dave Langford was saying: we patient serfs at Ansible Information have had it up to here with these people. We are sick of their pompous, protective attitude, and the fact that not a single hair on the blessed head of WordPerfect can be criticised. We are sick of the fact that neither we nor our customers can get any sensible Apricot technical support. We are sick of their bloody electronic telephone music, their erratic discount structure, the pious cracks from their superannuated employees about our unreasonable wish to turn an honest profit. We are sick of getting complaints from our customers about these people (more complaints, pro rata, than we've ever had about Apricot plc).

None of this would matter so much if the software worked as described. It didn't, and it was their responsibility. In the end we (or more properly, Dave Langford) did most of the repair work ourselves ... or as much as could be done without breaching WordPerfect Corp's copyright.

The fact that when we had done all this these same people then got on their high horse and accused us of passing off *our* work as *theirs*, and turned a big firm of lawyers loose on us to teach us a lesson, does leave a certain bitter taste in the mouth.

Mr Read has wandered into all this, apparently without noticing Dave Langford's very vexed expression. Even so, his comments require answers.

Script scraps

He seizes on the one admitted major shortcoming in WordPerfect: the problem associated with laying out a script. Fair comment, but he immediately sets free a whole shoal of red herrings by (a) misunderstanding the problem, (b) finding a 'solution' in SuperWriter that is (c) not only cumbersome and (d) probably unreliable, but (e) also needlessly complicated!

Let me try briefly to sort all this out.

The problem is not one of formatting in two columns in the manner shown in his article. That is simply done, in both SuperWriter and WordPerfect, by using (respectively) SuperWriter's CONTROL P h hanging indent or WordPerfect's CONTROL TAB — **>Indent<**. Mr Read's use of complicated embedded format commands tests a general rule of SuperWriter — that the more embedded commands there are knocking around, the more likely it is that SuperWriter will do something wrong.

The actual WordPerfect problem is illustrated by a a TV script with three columns. Each group of columns has to be glued together by Block Protect commands. The point to note is that different columns in the same *group* are of unequal length. Column 1 (camera directions) might be long, short or non-existent. Column 2 (characters' names) will usually consist of one line only. Column 3, the dialogue, can be long or short.

These columns are 'fixed' by Block Protect codes. This enables the writer to go back and revise the work — to make a piece of dialogue longer or shorter, or to insert a camera direction, or whatever. Doing this does not affect the overall structure of the columns; so if a foregoing group is expanded, those following are not mucked up.

So far so good. But should one of the columns grow so much that it reaches the end of the page (in other words, should a soft pagebreak be necessary) that column and its companion columns 'wrap' to the new page, because they are Block Protected. This makes text handling very cumbersome and unpredictable ... not something you want to deal with when writing.

Worse, should any of the columns grow so long that it exceeds the number of lines in a page, the whole column structure breaks down.

The programming is therefore flawed; it does not work as described. To use WordPerfect for scripts means working within limitations imposed by the software, which is not what you pay £425.00 for. For example, it is impossible to 'retrieve' an existing document into one column of a specially prepared new document. One of our customers wanted to do exactly this — converting previously-prepared text into a script for a documentary film — and wasted days trying to find out why page 2 and all pages thereafter were a crazy zigzag of columns.

This is an important flaw because WordPerfect's ability to work in columns is one of its selling points. (At least three of our customers bought WordPerfect for this feature.) As Dave Langford said in his article, we fully expected version 4.2 to correct the problem that we and others had complained about in 4.1. Instead, they changed the manual! "Parallel Columns are designed for documents such as scripts." Then the world-famous afterthought: "The Parallel Columns feature is designed for use with column groups that are shorter than one page".

Balancing act

Now, as Mr Read points out, you can get around this by starting out in a different frame of mind. In both SuperWriter and WordPerfect it's possible to set up multiple tab-stops, and create *de facto* columns by jabbing at the tab key or by using hanging indents. If the numerous keystrokes start being repetitive, you can write SuperWriter execute files or WordPerfect macros to minimise the boredom.

I've written scripts with both SuperWriter and WordPerfect using such methods. Both word processors were easy to set up and use, and both produced attractively-printed scripts. But given the choice, I'll always use WordPerfect in future.

I like the absence of visible print controls. I like seeing underlines and boldface text on the screen. I like having two documents in use at once, for ease of revision. I like being able to see pagebreaks. I like having the security of an instant 'undelete' key. I like the clarity, ease and precision of WordPerfect's block moves. I like the fact that the word-count is exact. I like being able to leave my document temporarily to work in MS-DOS and then return instantly to where I was. I like being able to number scenes automatically, and let the program worry about renumbering them if I later decide to move something around. I like the fact that should I wish to index my work, enter footnotes or endnotes, create a bibliography, include a few sums, sort part of the document into alphabetical or numerical order ...or any of these, then I can operate them with a minimum of fuss and without having to load some additional software.

In short, I like to write with WordPerfect. I'm obviously not alone in this: WordPerfect is now the bestselling word processor, and deservedly so. Mr Read has only to look through some back issues of APRICOT FILE to find several reviews that describe its coruscating wonderfulness more exhaustively than I have done.

But nothing is perfect. WordPerfect comes fairly close, but it is no more 'perfect' than SuperWriter is 'super'. Dave Langford and I have learnt to look at all proprietary software with a jaundiced eye, hoping for the best but ready for the worst. There's a lot of the former in WordPerfect, and hardly any of the latter. That's a good score, and any Apricot user who wants to get the best out of the machine should give it serious consideration. ◆

Free Small Ads

Apricot Sales

■■■ APRICOT XI 10MB hard disk, 720KB floppy, 9in monitor, 512KB memory, BIOS 2.7; usual Super software plus COMM, CARDBOX, utilities, etc. £925 plus VAT ono. Phone Peter Stamp, 0483 277703 (day) or 0483 277169 (evenings).

■■■ APRICOT XI 10MB hard disk, 720KB floppy, 12in monitor — £950. Also MICROLINE printer with Apricot/Sirius cable — £95. Call Mike on 061-428 5975.

■■■ APRICOT PC with twin 720KB drives, NEC V30 processor, 512KB memory, 9in green monitor; usual Super software plus LMI FORTH-83, Turbo Pascal, and nearly enough DEALS FOR READERS disks to set up in competition with TP Group ...£500 plus VAT. Call B Stanier, Demic Scientific, 0740 20818 any time.

■■■ APRICOT XI with 256KB RAM, 10MB hard disk, 720KB floppy, 9in monitor and some software — £900. Also APRICOT PC with 256KB, twin 720KB drives, 9in monitor and some software — £700. Both machines in excellent working order. All offers considered. Call 0934 33875.

■■■ APRICOT XI 768K RAM, 10MB hard disk, 720KB floppy, MS-DOS 3.2, SuperWriter, disks, cables etc — £750 plus VAT. Phone 0732 62396 9am-8pm any day.

Apricot Add-on Sales

■■■ INFRA-RED MICE for Apricot F Series, white; £40 each. POINT 32 network cards £99 pounds each. MANUALS available for PC/XI and F Series Apricots. Call Chris on 01-394 1766.

Other Sales

■■■ WATANABE WX4675 six-pen A3 flatbed plotter, little used. £450 plus VAT. Call B Stanier, Demic Scientific, 0740 20818 any time.

■■■ IBM CLONES supplies - PCs and ATs, 8-12MHz; prices from £750. Martin Filmer Computing Services. 01-761 0425.

Help

■■■ F SERIES user wants to get in touch with others in the States. Contact Bob Mandel, Lewis & Clark College, Box 96, Portland, OR 97219, USA.

■■■ PORTABLE user would like to contact another in South London to exchange information. Contact Geoff Reade, 9 Taymount Grange, London SE23 3UH.

Wants

■■■ IBM-COMPATIBILITY FOR F SERIES — either a fully-effective IBM emulator or a patch permitting the IBM version of Microsoft Word 3.0 to run. Also wanted — a WORLD MAP or GLOBAL MAPPING PROGRAM. I have scores of home-grown GW-BASIC programs to sell, exchange or share — including full documentation and sample programs for the infra-red Mouse. Contact Bob Mandel, Lewis & Clark College, Box 96, Portland, OR 97219, USA.

■■■ SECONDHAND OR EX-DEMO HARD DISK — internal or external, 10 or 20MB, for PC. Cannot afford £800+ for a new one. Write to T Orekoya, FAO (FIRI), Via della Terme di Caracalla, 00100 Rome, Italy.

■■■ MMS INSTRUCTION MANUALS AND INTERFACE BOARD required for MMS hard disk unit and tape streamer. Willing to pay for help (even telephone assistance) in setting this up. Call Roy Roberts, 0923 775381 (office) or 0494 775088 (evenings).

■■■ GOOD DEBUGGER wanted for PC/Xi. I hate using Apricot's; can anyone help before I'm forced to write my own? (Preferably with symbols and/or C source display). Phone 01-948 1603 any time.

■■■ 10/20MB EXTERNAL HARD DISK required for Apricot PC with BIOS 2.4. G S Jayaram, PO Box 2642, Doha, Qatar (phone 417282 6-9pm UK time).

■■■ APRICOT PC or 10MB XI wanted; also PRINTER if available. Fair price paid, immediate cash. Phone Mrs Morrison, 0636 704208.

■■■ WORD PROCESSOR wanted for Apricot PC with single-sided disks — MS-WORD 3.0 preferred. Also ANY OTHER SOFTWARE. Phone 01-908 4631

TP Group Services

■■■ DISK RECOVERY: problems with deleted files, scrambled hard disks, and the like? We can (probably) help — no fix, no fee! Call Roger Gann at TP Group on 01-833 3501.

■■■ PRINTER: we can get good prices on most matrix, daisywheel, inkjet and laser printers. Call Roger Gann at TP Group on 01-833 3501.

■■■ FILE TRANSFER: we can convert files between Apricot single- and double-sided, IBM PC format, PC/AT format, and Sirius — cheap! Call Roger Gann at TP Group on 01-833 3501.

■■■ TURNKEY CONSULTANCY: we can recommend and/or implement systems using popular packages or custom-written software. We can also convert you from Sirius or Apricot to another species. Modest prices, guaranteed work. Call Roger Gann at TP Group on 01-833 3501.

■■■ DESKTOP PUBLISHING: we can produce your newsletters, price lists, press releases etc in near-typeset-quality on our laser printer using our DTP software NEWSWRITER from your disks in virtually any format produced by word processor and/or computer. Fast turnaround, guaranteed quality, low prices. You're reading an example of our work right now ... Call Roger Gann at TP Group on 01-833 3501.

SERVICES

■■■ REPAIRS guaranteed. Martin Filmer Computing Services. 01-761 0425.

■■■ A0 PLOTTING — time on a Calcomp A0 plotter available for rent: Sirius or Apricot disks plotted on your paper or ours. Terms negotiable, but cheap. Phone James McEvoy on 01-859 4014 or 01-852 7981; or write to 16 Dunvegan Road, London SE19 1SA.

Use the service!

The Classified Ads service is free to all readers of APRICOT FILE. Just let us have a sheet of paper with the words (not too many); your ad will appear at least once in the newsletter and for around three months in the DEALS FOR READERS broadsheet.

Send your ad to us via electronic mail (Telecom Gold 81:JET767) or by land mail to:

APRICOT FILE
TP GROUP
PO Box 509
London N1 1YL.

The Whistling PC

Using the Apricot Sound Generator

By David Ingamells

Most likely the majority of users are not even aware that the Apricot PC can play tunes and even simple chords. You know that it generates electronic clicks and will beep in response to a CONTROL-G; but these are only the beginning of your machine's capabilities. With the programs listed here you can play your favourite melody — and even compose some of your own!

The Apricot PC/Xi has a sound generator that rates as 'sophisticated' in comparison to some rivals; it's a peripheral device that sits inside the main unit and consists of a few chips and a small loudspeaker.

It is accessed through the 8086's I/O space (more on this later). There are three independent sound channels, each capable of producing a note in the range 61.1Hz to 12.5KHz; and a noise generator (which can work independently, or can be driven by tone channel 3).

Channel selection

In normal use, channel 1 is used for the beep and the noise channel used for the key-click; and if . If these channels are changed the click and beep may sound different, or they may not sound at all.

The sound device needs no CPU control once it is triggered, so the program can proceed while the sound continues until another signal is sent to stop it. If a sound of set length is needed, though, the CPU will have to perform the timing — for instance, with the Turbo Pascal function `delay()`.

The device is programmed by sending it information in bytes with a special format. The following key is used in all the byte formats:

C2	C1	CHANNEL
0	0	channel 1
0	1	channel 2
1	0	channel 3
1	1	noise channel

Setting volume

One byte is sent which will set the volume of any of the four channels. The volume number is in the range 0 to 15, where 15 is off and 0 is maximum. The difference between each level is 2dB.

bit	7	6	5	4	3	2	1	0
	1	C2	C1	1	V3	V2	V1	V0

The lower four bits — V3 to V0 — form the volume number.

Setting the tone channel frequency

Two bytes need to be sent with a bit format containing the frequency and the channel number:

BYTE 1

bit	7	6	5	4	3	2	1	0
	1	C2	C1	0	F3	F2	F1	F0

[Note that C2,C1 = 1,1 is not used here]

BYTE 2

bit	7	6	5	4	3	2	1	0
	0	na	F9	F8	F7	F6	F5	F4

[Bit 6 is not used]

Bits F9 to F0 make up a ten-bit number (in the range 0 to 1023) which determines the frequency, using the equation ...

```
FREQ = 10*6 / (16*F)
```

If you use this equation on the two frequencies given earlier for the range, you'll find that 61.1Hz is tone number 1023 and 12.5kHz is tone 5. Tones 0 to 4 do not make an audible sound.

Setting the noise channel

The noise channel can produce either 'pink' or periodic noise, in the latter case with a set reference frequency or one determined by channel 3. One byte is sent to the sound device:

bit	7	6	5	4	3	2	1	0
	1	1	1	0	na	M	R1	R0

M sets the mode — 0 for periodic noise, 1 for pink noise. R1 and R0 set the reference frequency, as follows:

R1	R0	MEANING
0	0	3.90 kHz
0	1	1.95 kHz
1	0	975 Hz
1	1	use channel 3

Sending the control bytes

Peripheral devices are connected to the 8086 in what is termed the I/O space, and are communicated with by using the assembly language instruction OUT. Fortunately most high-level languages — including MS-Basic, Turbo Pascal and Zorland C — have a function to do this.

Each device connected to the I/O space has its own address(es) or ports; and for the sound device, the address is 50 hex. So all you need to do is to build up your control bytes and OUT them to address 50. To send 100 decimal to the sound generator, then, you might use one of these instructions:

```
Port[$50] = 100;                    [TURBO PASCAL]
Out &H50,100                        [MS-BASIC]
outp(0x50,100);                     [ZORLAND C]
```

Music

Now that we know how to tell the sound device what to do, we need a bit of music theory to make *meaningful* sounds.

■ Notes

Musical notes are grouped into octaves of 12 semitones. The semitones are named in sequence ...

A A# B C C# D D# E F F# G G#

... where that # means 'sharp'. (To complicate matters, A# can also be called Bb where *b* means 'flat'; but we will not be using flats here.) Once you reach G# the next note is again A, but one octave higher than the previous A.

So what is the difference between A and the A one octave above? Well, the higher A has double the frequency of the first one. The semitones are divided up evenly between the two As; and one semitone is calculated from the next by multiplying the frequency by the twelfth root of 2 (1.059463094). Now all we need to know to calculate the frequency of any note is a reference frequency. The internationally recognised standard pitch is A = 440 Hz (and hence 110, 220 etc).

■ Rhythm

A musical tune is divided into bars, each holding a fixed number of beats which is determined by a time signature. A very common signature is 4/4 time — meaning there are four quarter-note beats (crotchets) in each bar. The length of each beat is indicated by a count per minute; but this is often shown by composers only by a vague term such as *andante* and it's then left to the judgment of the musician.

A note can have a length of 1, 1/2, 1/3, 1/4, 2, 3, 4 or more beats, the only rule being that the sum of the lengths of the notes in a bar must be the length of the bar as given in the signature.

The programs

I have divided the tasks of making music into [1] encoding the tune and saving it on disk; and [2] playing the tune.

MUSBUILD.PAS is the program given the first task. Before you run it, examine the music and find the shortest note that will allow you to give all other notes a length which is a whole multiple of this one's length. Then go through the piece and pencil in the name of each note and its length as a multiple of the shortest note.

So that each note can be entered as a single letter, the sharps are represented by the same letter but in upper case. As a guide to the octaves, middle C is in octave 3; but there is no need to keep to this. The tune will still sound the same in another octave — it'll just be higher or lower.

As you work through the piece you may find that you will want one channel to sound on while another changes. To do this, type in 'n' for the note name in the channel that is to sound on ('n' means no change). If you want a channel to stop, enter a zero volume. To end input, type 'z' for a note's name.

You will then be given a chance to correct any note. Type in the notes for a chord just as you would the first time. If you don't want to enter the whole piece in one go you can stop at any time, play what you have already entered, and then extend the piece later — just answer 'n' to the prompt asking whether the piece is new.

To play the tune, use MUSPLAY.PAS. The only input is the name of the tune file and a number of beats per minute — experiment with this number to find a good speed.

MUSBUILD.PAS

```
Program MusicBuilder;
{ author : D. Ingamells  6th Sept 1987 }

type
notearray = array[1..3] of integer;
Octavetype=0..6;
Volumetype = 0..15;
volumearray = array[1..3] of volumetype;
chordtype = record
note:notearray;
vol:volumearray;
count:integer;
end;
tune = file of chordtype;

Const
a0 = 55;
numKey = 12;
twelth_root_2 = 1.059463;
del = 0;
{*********************************************************************}
{ power — raise a number to a positive integer power }
{ num — the number to raise }
{ raised — the power to raise it to }
{ note that this could be done with logs }
{*********************************************************************}
Function Power(Num:real; Raised:Integer) :real;
var
i :integer;
p : real;
begin
%p := 1;
%for I:= 1 to Raised do
%begin
)P := P * Num;
%end;
%power := p;
end;

{*********************************************************************}
{ freq — derive the frequency of a note }
{ note — number of note "a" = 1, "g#" = 12 }
{ octave — 0 gives "a" = 55 Hz }
{*********************************************************************}
Function Freq(note:integer;octave:octavetype) :real;
begin
Freq := (1 shl octave) * a0 *power(twelth_root_2,note);
end;

{*****************************************************************}
{ sound_num — convert a frequency into the representation }
{ required by the sound device }
{ freq — frequency to convert }
{*****************************************************************}
function sound_num(freq:real) : integer;
begin
sound_num:=round((1000000.0 /16.0)/freq);
end;

{*****************************************************************}
{ get_chord — ask user for the values that make a chord }
```

```
{ converting note and octave to a number }
{ also handles termination: }
{ chord — record holding values for each sound channel }
{ sound number and volume plus length of chord }
{ iret — = 0 for normal input }
{(= 1 if user types x (eXit) for a note name }
{*****************************************************************}
procedure get_chord(var chord:chordtype;var iret:integer);
var
i":integer;
notename : char;
oct":integer;
vol":integer;

begin
with chord do
begin
for i:= 1 to 3 do
begin
write('note for channel ',i,'? ');
readln(notename);
if notename = 'x' then
begin
writeln('exit selected');
iret := 1;
exit;
end
else if notename = 'n' then
begin
note[i] := 0;
vol[i] := 0;
end
else
begin
case notename of
'a' :)note[i] := 1;#{ note 'a'}
'A':)note[i] := 2;${ note 'a#'}
'b':)note[i] := 3;
'c':)note[i] := 4;
'C':)note[i] := 5;
'd':)note[i] := 6;
'D':)note[i] := 7;
'e':)note[i] := 8;
'f':)note[i] := 9;
'F':)note[i] := 10;
'g':)note[i] := 11;
'G':)note[i] := 12;
end;
write('octave for channel ?');
readln(oct);
write('volume for channel ?');
readln(vol[i]);
{convert note to a sound number }
note[i] := sound_num(freq(note[i],oct));
end;
end;

write('number of beats? ');
readln(count);
iret := 0;
end;
end;

var
```

```
score: tune;
name :string[66];
chord : chordtype;
i, iret : integer;
reply: string[3];
begin
write('enter name for tune file? ');
readln(name);
assign(score, name);
write('is this a new file?');
readln(reply);
if reply = 'y' then
begin
{ build a new tune }
rewrite(score);
writeln('Enter note name, octave(0-6) and volume(0-15) for each of the ');
writeln('3 channels (e.g. "C 2 8") c = "c", C = "c sharp"');
writeln(' for note name "n"= continue note from previous chord,"x"=done');
writeln('Then enter number of beats before next chord is to start');
get_chord(chord,iret);
while iret = 0 do
begin
write(score,chord);
get_chord(chord,iret);
end;
end
else
begin
{extend an old tune}
reset(score);
write('do you wish to extend the file?');
readln(reply);
if reply = 'y' then
begin
seek(score,filesize(score));
get_chord(chord,iret);
while iret = 0 do
begin
write(score,chord);
get_chord(chord,iret);
end;
end;
end;

{ chance to correct any wrongly input notes }
write('do you wish to correct any notes?');
readln(reply);
if reply = 'y' then
begin
{ first chord in file is number 1}
write('chord number?');
readln(i);
while i > 0 do
begin
{replace note i}
seek(score, i-1);{ turbo counts records from 0 }
get_chord(chord,iret);
if iret = 0 then write(score,chord);
write('note number?');
readln(i);
end;
end;
close(score);
end.
```

MUSPLAY.PAS

```
Program MusicPlayer;
{ Author:  D Ingamells  6th Sept 1987 }
type
Volumetype = 0..15;
notearray = array[1..3] of integer;
volumearray = array[1..3] of volumetype;
chordtype = record
note:notearray;
vol:volumearray;
count:integer;
end;
tune = file of chordtype;
{*************************************************************************}
{ play_note(channel,note,volume) }
{ channel — sound channel to play note on }
{ note — note number from 1 to 12 }
{ volume — volume of note }
{ note that the note continues to sound until another play_note call }
{ switch off with a zero volume }
{*************************************************************************}
procedure play_note(channel,note:integer;
volume:volumetype);
var
outbyte:byte;
begin
if ((note > 1023) or (note < 13)) then
begin
volume := 0;
end;
{set old note to zero}
outbyte:= $80 + (32* (channel-1)+16 )
9 + 15 ;
Port[$50]:= outbyte;
delay(10);
{set new freq}
outbyte:= $80 + ((32*(channel-1))) + (note AND $000F);
Port[$50]:= outbyte;
outbyte:= note shr 4;
Port[$50]:= outbyte;
{set new volume}
outbyte:= $80"+ (32* (channel-1)+16 )
9 + ((15 - volume) AND $000F);
Port[$50]:= outbyte;
end;

var
reply:string[4];
beats: integer;
i : integer;
score: tune;
name :string[66];
chord : chordtype;
begin
write('enter name of tune file? ');
readln(name);
{ more beats means a quicker tune — try 100-200 as a starter}
repeat
begin
'write('Number of beats per minute?');
'readln(beats);
'{ calculate length in milliseconds of each beat }
'beats := (1000 div beats) *60;
'assign(score, name);
'reset(score);
'repeat
'begin
read(score,chord);
with chord do
begin
{sound the chord...}
for i:= 1 to 3 do
begin
{ ...by sounding each channel's note}
if note[i] > 0 then play_note(i, note[i], vol[i]);
end;
delay(beats*count);
end;
end;
until eof(score);
{silence the three channels }
play_note(1, 3, 0);
play_note(2, 3, 0);
play_note(3, 3, 0);
write('again? ');
readln(reply);
end;
until reply = 'n';
end.
```

This method of control will work with all BIOS and MS-DOS versions, as the control is going direct to the device. With BIOS 3, however, the Control Device has all the functions needed to use channels 2 and 3:

Sound Driver: device 38 Hex

Command	Description	Data
03	select channel 2 volume	volume
04	select channel 3 volume	volume
05	select channel 2 frequency	frequency
06	select channel 3 frequency	frequency

In each case, RET (AX register) will return the current setting. It would be advisable not to mix the two methods of control, and to use the BIOS where possible.

A tune

This tune only uses one sound channel, so enter 'n' for channels 2 and 3. It uses two octaves, but there is no requirement that it start in any particular octave; so choose one yourself — the lower ones sound better) Where the higher octave is used it is indicated by +1 after the note name.

The Proms and beyond

Now that you can make sounds, there are all sorts of possible applications for the sound device. You could write a real-time synthesiser with the keyboard acting as a piano — it is possible to shape the notes so that they vary in volume or pitch (for vibrato, for instance) although the device is fairly slow and will not respond to the quickest changes. Games can be made to sound like their arcade versions. On a more simple level, interactive programs could have a range of different beeps to indicate errors or various input requests (very useful if, like me, you need to look at the keyboard while you type!). ◆

Note, octave	length
d	3
d	1
e	4
d	4
g	4
F (sharp)	8
d	3
d	1
e	4
d	4
a+1	4
g	8
d	3
d	1
d+1	4
b+1	4
g	4
F (sharp)	4
e	4
c+1	3
c+1	1
b+1	4
g	4
a+1	4
g	8

The Easy Way

The two programs here will of course be made available on a forthcoming DEALS FOR READERS disk — watch this space!

Contributors in this issue

Dennis Jarrett is Editor of the newsletter. His magazine credits include being founder-editor of *Practical Computing* and *Which Computer*. Among his books are *The IBM PC Book*, *The Sirius/Victor 9000 Book*, and *The Good Computing Guide* (very cheap, at £3.00 inc P&P from DEALS FOR READERS).

Edward Bromhead is Reader in Geotechnical Engineering at Kingston Poly, where he uses a couple of Apricots to model the behaviour of soil slopes and groundwater (mainly using his own Fortran software and GSX). He specialises in earthworks (eg for motorways) and landslides (he predicted the collapse of Carsington dam in November 1983: it fell over eight months later).

Roger Gann works full-time running the DEALS FOR READERS section of the newsletter as well as contributing words, ideas, legal advice, etc. He was formerly a builder, a lawyer, and a kind of grandiose bank clerk.

Terence Green is an unemployed Apricot owner (currently looking for a job). Formerly an unlicensed truck driver in Lesotho, he started on IBM 370s and worked his way down to micros; these days he minds the kids and scrapes a living washing dishes (and writing occasionally for APRICOT FILE).

David Ingamells is an analyst/programmer working on a VAX programming a database/graphics/scientific package for the oil exploration industry. His employer uses Apricots to talk to its old (early 1970s) computers; he bought an Apricot to get into music synthesising "and am still trying, three years later".

David Langford is a full-time author and computer addict, whose latest book *EARTHDOOM!* — a spoof of disaster novels, written with John Grant and scorned by Kingsley Amis — is "by no means" about Apricot BIOS variants. (The inclusion of a permanently drunk character called Apricot should not be misinterpreted.) He is inordinately proud of being a guest of honour at the 45th World SF Convention last August.

Christopher Priest is the second half of Ansible Information — the first half being APRICOT FILE's columnist David Langford. In real life he is another novelist (latest book: *The Glamour*, published by Abacus — free plug); he wandered accidentally into the world of computer software, and doesn't regret it at all. Sometimes he dreams about wandering out again and writing another book.

Mark Whitehorn started life as a Cytogenicist but the Siren Song of the computer was too strong and he is now working as a programmer at a Scottish university, particularly on the application of computers (especially Apricots) to education. He says his hobbies include penguins, building 1960s racing cars on the cheap, and indoor underwater powerboat-snooker (of which only the last is apparently untrue).

A Moving Tale

Changing from GSX to GEMVDI

By Eddie Bromhead

For most people, moving house is a move for the better. It's normally a painful business, and costly too; but hopefully the benefits in terms of a better environment and improved facilities will soon show up.

Statistics show that moving is one of life's traumas. Perhaps the week of moving house isn't the time to change simultaneously to a new graphics system on my Apricot. Neither seemed a big move when I first contemplated them: I just hope that the house is nowhere near the nightmare of changing from GSX to GEM ...

The reasons for deciding to change are a little varied. First, Amstrads are becoming commonplace and they offer GEM graphics as standard. GSX drivers do exist for some of the IBM graphics systems (notably Hercules and CGA); but they are primitive. On the other hand, there are GEM drivers for virtually all IBM graphics systems, and they're much more advanced.

Second, GEM graphics offers a number of features not available under GSX. These include laser printer drivers, greatly improved text facilities, more and better 'area fill' patterns, and high-resolution support for the C Itoh printers that ACT used to own-badge. So there are advantages not only when porting to IBM-compatibles but also for running graphics applications on Apricots.

On the principle of there being no such thing as a free lunch, a price has to be paid for high-resolution dot matrix printer graphics. Unless you use decent quality paper, the pounding it takes from all those pin impressions makes it stretch locally — and paper of decent quality isn't cheap.

The graphics system which underlies GEM, the so-called VDI, is an evolution from the GSX system supplied with most Apricots. GEM also has a high-level system called the AES which manages the overlapping windows, drop-down menus and so on. It may not be widely appreciated, but it is possible to utilise the VDI in a program without necessarily taking on board the whole WIMPs razmatazz. Indeed, making a WIMPs program out of an existing application — even one written to use some sort of graphical interface — will probably call for a complete re-write.

Some of the experience in redeveloping these programs is condensed in the following account. I do have to confess, however, that the conversion process is still incomplete. Advice from those more knowledgeable is welcomed!

Loading GEMVDI

Both GEMVDI and GSX use the ASSIGN.SYS file to associate 'devices' with 'device numbers'. In GSX, the driver with the largest size in bytes comes first in ASSIGN.SYS. This is because when GSX is invoked, the first named driver file is loaded into memory; and further drivers use the same RAM if they are loaded at a later stage. The biggest has to be loaded first so that room for any other is reserved.

In contrast, GEMVDI assumes that the screen driver will remain in memory at all times; driver number 01 is flagged with a little 'r' to denote that it is resident. Other drivers are loaded elsewhere in memory as required.

The second difference is that GSX loads and stays resident when the 'GRAPHICS' command is given. If the 'GEMVDI' command is given, it loads the GEM Desktop. To use GEMVDI on its own, the application name is given on the command which invokes it:

```
A> GEMVDI /programname
```

This loads GEMVDI, runs the named application, and unloads GEMVDI on completion. On a hard disk machine, with GEMVDI in the GEMSYS subdirectory, it also returns control to the root directory whether you want to or not.

Also in the GEMVDI's ASSIGN.SYS file are the names of the alternative, loadable, character fonts. Unlike GSX (on the Apricot!) there is only one system font in the screen driver, although it is possible to load alternate font bit maps. This will not normally be needed in the first stage of conversion from GSX to GEMVDI if screen graphics alone are required.

Calling GEMVDI

GEMVDI is called from a compiled language via an identical method to calling GSX. I have used both Harrison-Ward Associates' GSX bindings and the simple interface from MS-FORTRAN which was supplied by Apricot. GEM recognises the GSX call, although it (officially) requires interrupt EF rather than GSX's E0h. The cognoscenti will all now rush to DEBUG and search their favourite bindings for CD E0 sequences to change to CD EF!

(Hint: if you want to call the GEM AES as well, DEBUG any GEM application and look for CD EF. This crops up in two places, one of which you will recognise from GSX, and which is the VDI call — the other is the AES call. The AES expects a parameter block which differs from that needed for the VDI).

All the familiar arrays are required, except that CONTRL is one element longer. CONTRL(7) and

INTIN(11) reflect the major changes. INTIN(11) must be specified at OPEN WORKSTATION; it may have the value 0 or 2. The value 0 makes GEMVDI use the dear old NDC system, using 2 specifies a 'real coordinate system' (with its origin at the upper left of the screen, printer or whatever). The coordinates on a screen device are pixels, on a printer they are dots, and on a plotter they are plotter steps. Use NDC for total compatibility with GSX, or if you want the software to run on machines with different screens.

CONTRL(7) is another departure. GEMVDI passes back from the OPEN WORKSTATION call a number called the Workstation Handle. Future operations must specify this Handle to perform graphics on the right device — remember, more than one device can be open simultaneously under GEMVDI. After opening the screen workstation, save its handle.

GEMVDI Functions

In the main, these are an improvement on GSX functions. There are more primitives, there is 'clipping', and (on non-Apricots) the text operations are much more elegant.

In the area of Input, however, GEMVDI is much less easy to use because it operates at a lower level than GSX, expecting high-level operations to be done by the AES.

For example, in GSX there is a function called 'Input Locator'. This tracks the cursor, which may be moved about with cursor keys or mouse, and returns the coordinates when either the mouse button or the Enter key is pressed. Despite the documentation, the 'Input locator' function has been deleted from GEMVDI. Instead, you have to do this:

Operation	Opcode
1. Define the shape of the cursor	111
2. Show the cursor	122
3. Track the cursor, sampling the button state	124
4. Repeat step 3 until the state changes	
5. Hide the cursor	123

[Opcodes numbered above 100 are innovations for GEMVDI]

This does not check the keyboard for input — that has to be done with separate code. My first response is to assume mouse input *only*; this isn't a problem when porting graphics applications to the Amstrad, but might be on other machines.

Step 3 in the table is a 'real time' operation. Unlike the GSX Input Locator, it is not possible to 'click ahead' in the same way as you 'type ahead' of the screen response. This isn't too bad if you've ever typed or clicked ahead and found yourself confirming the deletion of an important file, but it can be tedious if using a slow machine. But when inputting a stream of coordinates, especially on a fast Xen, you'll find that when the loop of steps 3 and 4 is called into play a second time to get the next point(s) it will repeatedly catch the same click!

So some logic must be inserted to check that the mouse button has been released and pressed again. I've also had to include timewasting loops, as attempts to measure passage of time by calling the MS-DOS time routine seem to hang GEM.

Those poor souls whose Apricots arrived with ACTIVITY can find details of the calls for these operations in a text file called GSX.DOC on their issue disk; this also details many other opcodes for GEMVDI enhancements which Apricot put into later GSX driver releases, those numbered above 100. If you who can't find this file, buy a book — not Balma and Fitler (see my review in APRICOT FILE Release 2.9), nor the Digital Research manual, but instead *GEM on the Atari ST* by Norbert Szczepanowski and Bernd Gunther (by First Publishing out of Data Becker at £12.95). At first sight this may seem an odd choice; but it does give all the call sequences — though it repeats the nonsense that the Input Locator call works, and of course it keeps referring to Atari's TOS operating system. This book has been translated from the German, but still comes out about ten times as readable as the source manual. (This is more a criticism of the original than praise of this book, so don't say you weren't warned. It also contains much that is inapplicable, and some things that are demonstrably wrong — notably the bits most obviously plagiarised from the DR documentation).

The function which samples the mouse button(s) also tracks the cursor, returning the cursor position at each call. When sampling keyboard input, however, you need to decipher the keys, find out if a cursor arrow key has been pressed, sample the state of the SHIFT, CONTROL, and ALT keys, and move the cursor accordingly — keeping track of whether slow or fast motion is in use, whether no key has been pressed, and so on. To make a general program, this needs to be interleaved with the sampling of the mouse button state. Come back GSX — all is forgiven!

This is my sequence:

Operation	Opcode
1. Define the shape of the cursor, set internal flags (eg speed of movement)	[OPCODE 111]
2. Show the cursor	[OPCODE 122]
3. Sample the state of CTRL, SHIFT, and ALT	[OPCODE 128]
4. Input string, sample mode, sampling one character (set number of characters to -1 to obtain GEM-standard keyboard. This is correct EXCEPT for the detection of SHIFTs — important on cursor keys)	[OPCODE 31]
5. Decipher key(s) as follows:	
no change	repeat from step 3
movement	update position, hide cursor, repeat from step 2
change	reset internal flags, repeat from step 3 option key
terminate	hide cursor, return current position

Interleaving the two methods of cursor tracking is even more tedious.

When building a cursor, two blocks of 16x16 bits (each 16 words) need to be defined. The first of these is the MASK, the second the FORM. When the MASK is passed over a screen image, those pixels set to 0 in the MASK allow the original image to show through: those set to 1 are temporarily turned to the background colour.

The FORM fits exactly over the MASK, and has a similar logic. Pixels marked 0 in the FORM show through — this applies whether they are from the original image, or those that are set by the MASK — and those designated with a 1 show as the foreground colour. In order that the cursor FORM should not be lost when the cursor moves into a block of foreground colour, the MASK shape should be slightly larger than that in the FORM. All this is illustrated in the diagram.

MASK	FORM
INTIN (6 ... 21)	INTIN (22 ... 37)
0000000000000000	0000000000000000
0000000000000000	0000000000000000
0000000000111111	0000000000000000
0000000001111110	0000000000111100
0000000011111100	0000000001111000
0111000111111000	0000000011110000
0111001111110000	0010000111100000
0111111111100000	0011001111000000
0111111111111110	0011111110000000
0111111111111100	0011111111111000
0111111111111000	0011111111100000
0111111111100000	0011111110000000
0111111110000000	0011111000000000
0111111000000000	0011100000000000
0111100000000000	0010000000000000
0110000000000000	0000000000000000

In addition, it is necessary to set colour choices and the position on the mouse FORM corresponding to the 'hot spot' or sampling position. This is done via the parameters of INTIN:

INTIN (1) x coordinate of hot spot (relative to top left corner)
(2) y coordinate of hot spot (relative to top left corner and zero-based)
(3) colour choices
(4) colour choices
(5) colour choices

To make the arrow function above, these five parameters need to be 2, 14, 1, 0, 1. On a black-and-green monitor, the arrow appears green, but when it moves over a green area, a black outline surrounds it.

Hardcopy

One of the advantages of GEMVDI over GSX is the sophistication and range of graphics hardcopy drivers. These include laser printers. This however is one area where GSX program logic needs to be changed. Instead of closing the screen workstation before opening a hardcopy workstation, *the screen workstation must be left open and the hardcopy device opened as well.*

Both GSX and GEMVDI have escape operations for switching between text and graphics screens. This is not so useful under GSX, where a 'close (screen) workstation' is just as effective. Under GEMVDI, I have found that closing the screen workstation can lead to unforeseen side effects. Indeed, before opening a hardcopy workstation, it is necessary to have the screen workstation in graphics mode. So much for all those messages — in text — keeping the user informed of progress on the printout that you put into your GSX program!

I am using GEMVDI on a 800x400 resolution Xen and a PC. The hardcopy drivers for this system do not provide the system font which the documentation says is provided for *every* driver. To obtain text, the 'load fonts' command (GEMVDI opcode 119) must be used. This in turn takes still more RAM; no wonder GEM needs to be run on a machine with a generous allowance of memory. After a 'load fonts' command, however, any request for the system font (number 1) gives 'Swiss' style or non-Serif text (literati will recognise the common origins of 'Helvetica' and 'Swiss').

Unlike GSX, which takes a single bit pattern and scales it up to meet all character heights, GEMVDI uses different patterns for different height ranges. In addition, the GEMVDI Swiss characters are proportionally-spaced and look very elegant in the larger sizes. On the screen, the effect is similarly impressive — though for tiny text the scalable Apricot GSX 'stroke' font is somewhat more legible.

GEMVDI also has methods of aligning text, of specifying character sizes in absolute terms, and a special function (actually one of the Generalised Drawing Primitives or GDPs) to spread or compress a string of text into a particular space. GEMVDI has functions for finding out all sorts of things: one of the more useful is to explore the size of the area which will be covered by a text string, which is difficult to compute since most fonts are proportionally spaced.

(Just in case you are tempted to lift the fonts for screen and printer from your Amstrad — it has 'Dutch' or Serified characters too — don't bother. They are pretty scruffy and of low resolution).

If you are prepared to make more effort, GEMVDI offers more patterns, more graphics primitives, and best of all, a rectangular 'clip' function to confine drawing to a particular area of the screen or page.

The picture sizes generated by GEM, larger than those produced by the Black Hole GSX drivers, correspond closely to the later (IBM) GSX drivers. Any software which assumed the page size needs to be modified to take account of this. On the other

hand, users of the C Itoh printers (mugs often bought these at the same time as their Apricot or Sirius in the early days — mine was a deliberate choice because of its Greek letters) will be delighted with the high resolution possible in comparison with the available GSX driver.

Not only are the range of area fill patterns and line types enhanced, but 'polylines' may be finished off square ended, rounded, or arrow headed.

Problem areas

As mentioned, graphical input is a problem area. Note that the slightest incursion into the AES is not only mind-blowing, but requires taking on board the whole of the GEM graphical system. This proved beyond me. Using the GSX input procedures will either cause the computer to hang or to proceed without stopping, depending on how you handle the coordinates which have been 'input'.

GSX is tidy-minded and space-conscious, so only one workstation needs to be open at a time (or for that matter is allowed to be). GEMVDI needs the screen workstation open *and* in graphics mode to do hardcopy. This may need some thinking about.

Despite the documentation, printer drivers don't have a system font: the 'load fonts' command is essential if text is to be output. On the printer, this sets both fonts 1 and 2 to the loaded style. It is possible to load extra fonts for the screen too. But rotating characters from an x-axis baseline to a y-axis baseline generates unreadable smudges on the PC/Xi/Xen screen for some character sizes because of the resolution.

Although I can access the different polymarkers in my programs, I seem unable to change their scale. Identical code in the GSX equivalent programs works perfectly. What am I doing wrong? I seem to have overcome this in part by using circle and bar primitives of small size instead, and/or by generating symbols with polylines.

Keyboard input is a nightmare. I think GEM loads a different keyboard. This is *not* the same as the one documented by DR, and definitely differs from the normal Apricot keyboard. Crash GEM in graphics mode, and you may need to re-boot by switching off completely (the normal CTRL-CLEAR combination which restores alpha screen mode after a GSX crash has been re-assigned to something else). This keyboard is obviously loaded via a BIOS 3.1 speciality, which is why GEM doesn't work with BIOS 1 or 2; and I suspect the GEM drivers use INT EC screen graphics.

If you access GEMVDI via the GSX interrupt, the OPEN WORKSTATION call works — even when the program is called up from the GEM desktop. But try it with the proper GEMVDI interrupt, and the system hangs! Obviously, the screen workstation is open already, and so the correct call might be to open a 'virtual workstation'. There may therefore be some merit in using the GSX interrupt if the program is expected to be called up from GEM desktop.

Finally, some GEMs are more equal than others. Amstrad PC1512 GEM is version 2.0, Apricot PC/Xi GEM is 1.2 — and is not being further developed either. Version 2.0 obviously needs something that 1.2 doesn't, because I can't get hardcopy from it. As an aside, Ventura Publisher needs GEM 2.2, uses IBM BIOS interrupts in its installation programs, and generally behaves badly on a real Apricot — meaning it crashes everything. So much for GEM promoting program portability!.

Last words

A GSX application ought to translate to a GEMVDI application with tiny alterations. These are primarily associated with input, where GSX is complete and GEMVDI is more primitive: and text output, where GSX is primitive and GEMVDI more sophisticated. The first of these is because GEM expects input to come via the AES, whereas GSX was expected to stand alone. Apricot GSX applications actually port more easily to IBM-style machines (especially the Amstrad) if they remain GSX-based!

If you thought GSX documentation was difficult to get, hard to read, and generally unhelpful, you hadn't met GEM. The available material is all of these things, and often wrong to boot. The conversion ought to be simple and painless, but it is neither.

The ultimate easy-to-convert software uses only these commands ...

- *initialise*
- *select pen*
- *move* [or *pen up*, *pen down*, and *move*]
- *draw* [or *pen up*, *pen down*, and *draw*]
- *wind-up* on completion

... and drives every other effect by high-level code. That much is easy in GEMVDI as it is in GSX. What a shame I was seduced from this pure state by easy-to-program Japanese plotters and even by GSX itself.

In conclusion — make the move only if you need one or more of these:

- the enhanced text
- laser printer support
- the ability to port programs to IBM-type machines when you cannot get hold of the necessary GSX drivers
- high resolution on an ACT Writer
- nothing more than vectors
- lots of aggro! ◆

ASD updates

The two original **Ansible** SuperWriter disks (ASD1 and ASD2) now exist in one volume only, simply called 'Ansible SuperWriter Disk'. **Chris Priest** of Ansible tells us the new disk includes all the best stuff from the two volumes "plus two or three new items, less the dross and the duplications". The revised ASD contains a total of some 91 files. Our new DEALS FOR READERS reference is AS060, the price is £86.25 inc VAT and P&P.

The Ansible WordPerfect Disk has also been discontinued, to be replaced by something Ansible calls PARAGON. It's identical in all but name; that results from an entertaining correspondence between Ansible and various lawyers about the use of a trademarked name — the only approved alternative name for the disk, according to the lawyers, would be 'Ansible Disk for use with the WordPerfect Software'. Punchy, eh? Price is unchanged at £57.50; "contents the same as before, but better" says Chris Priest. We have PARAGON in the DEALS FOR READERS lists at that price as AS061.

Old bill

Our Telecom Gold mailbox went dead for a week earlier this month. We checked, and we had paid the bill; so we called Telecom Gold to find out why our *ID* had been invalidated. "Ah" said The Man. "You have an 81:JET number." True, true. "And 81:JET numbers are part of Apricot's Micromail facility" (which runs as a kind of closed user group on Telecom Gold). "Well, it's not *you* that hasn't paid *your* bill — it's Apricot, which hasn't paid its bill for Micromail". Can this be true? I think we should be told.

Star expands

Star Computer Group, the USM-quoted group that is probably the UK's leading supplier of purpose-built systems for accountancy, has hitherto been a Unix specialist. It's now spreading its wings with the formation of micro products division based on the recent acquisition of **Orchard Business Systems** and Orchard's MS-DOS ledger packages. Third-party products will also be offered.

Brikat benefits

A year of consolidation paid off for **Pegasus** supplier the **Brikat Group** with profits more than doubled at **£2.9 million** while sales were substantially lower — down from £13.6 million a year ago to **£10.3 million** now. The fall in turnover results from divestment of Brikat's troublesome venture into retailing; all 15 of its Brikat Business Centres were sold off during the year. A name-change to Pegasus Group is planned, which should clarify things for everyone. Meanwhile Brikat's USM price hovers around 300p; the year-ago price was a quid higher.

Data Protection Act in force

"From 11 November we will have 40 million investigators" said **Data Protection Registrar Eric Howe**, who reckons that only half the probable candidate had in fact registered their data by the date on which the Act came fully into force. For holding personal data without registering, there are fines of up to **£2,000** in a magistrates' court (and with no limit in a higher court). The deal is that individuals can now seek access to data held on registered databases; data-holders can charge up to £10 per check, however, which led our contributor **Dermod Quirke** to come with a Brilliant Wheeze for making money — you register a database called something like 'The Personal Information Database', put absolutely nothing in it, print off a couple of thousand form letters saying "sorry we haven't got any data on you", and wait for the £10 cheques to start rolling in.

Excel for Apricot

Continuing a longstanding relationship of mutual enthusiasm, **Apricot** is the first **Microsoft** OEM to take the well-reviewed Windows-based **Excel** spreadsheet — modestly described by a Microsoft man as "the most significant product since VisiCalc and Lotus 1-2-3 opened up the micro market to spreadsheets".

AT Computerworld reprise

The newly-independent AT Computerworld has opened its fourth store to purvey products from its former owners Tandy and Apricot. The new Reading centre joins London, Bristol and Coventry; AT Computerworld says it's looking for more expansion.

Aids on Apricot

The Health Education Authority is setting up a database of treatment information about AIDS, using the Sculptor 4GL package and a multi-user Apricot network. Systems house Interdata supplied the computing; Sculptor was selected partly because proprietary DBMS systems couldn't handle the amount of cross-referencing requires, partly because it allows compatibility with Xenix and imported C programs. When the system is completed, interested parties — doctors, researchers, medical bodies — will be able to call HEA for an instant run-down of relevant information. ◆

Christmas is coming

If you're looking for Christmas gifts, consider a set of limited-edition prints by Terrance Duggan. Terry is the Database Manager here at TP Group, but he's also a professional artist with a keen eye for the kind of detail most of us miss. Over the last couple of years he has been looking at London, and from more than 300 works that resulted we have selected five groups of four pen-and-ink drawings to present as a Limited Edition of signed and numbered prints on art paper.

With this issue you should receive a brochure for the *Another London* series. The offer is exclusive to our newsletter's subscribers at a very special price; we feel that the prints would look well in any home, and they make unusual and unexpected gifts for Christmas — we can despatch direct to anywhere in the world, enclosing a Gift Certificate if required.

But don't delay — the edition is limited to just 250 sets of the prints, and we're already shipping them. And for Christmas despatch we *must* have your orders before 16 December ...