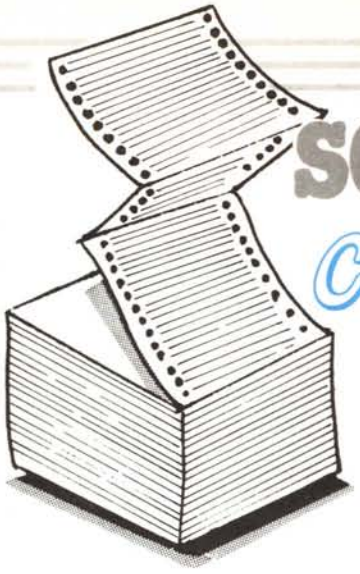

software C-64

a cura di Tommaso Pantuso

Geo Convert v1.2

di Daniele Finocchiaro
Linguaglona (CT)

L'uscita sul mercato del Geos ha dato al 64, che ormai si avvia verso la senilità, un soffio di vita. Se è vero che seguiranno vari applicativi sotto Geos, il 64 potrà vivere ancora a lungo! Tuttavia la non-compatibilità con tutti gli altri programmi per 64 rende estremamente chiuso questo mondo Geos. Ricordo che, dopo i primi giorni di entusiasmo per il Geos, avevo finito per accantonarlo, dato che per scrivere testi «seri» preferivo, e preferisco tuttora, affidarmi al SuperScript, e per disegnare al Doodle II. Che fare? Il Geos era un gioiellino, ma troppo isolato dagli altri tesori per 64. Cercai allora di capire come erano organizzati quei maledetti file USR (gli unici che il Geos tratta), e di scrivere un programma che trasformasse i file sequenziali di SuperScript (o EasyScript) in quelli user del Geos (dato che io utilizzo prevalentemente word processor). Il programma che presento è il risultato di questo «studio» dei file Geos.

Descrivere cosa fa il programma è quanto mai semplice: trasforma un file formato SuperScript in un altro formato GeoWrite. Più complesso è dire come lo fa. Dobbiamo innanzitutto analizzare, almeno sommariamente, la struttura dei file dei due programmi.

Il SuperScript si limita, da bravo ragazzo, a mettere tutti i caratteri battuti in fila, formando appunto, un file SEQ. I codici corrispondono a quelli ASCII. Uniche eccezioni sono i comandi, quelli cioè che sullo schermo compaiono in reverse: asterischi, che precedono altri comandi, comandi di sottolineatura, etc. Nel file viene memorizzato anzitutto un carattere $80 (dec 128), poi i caratteri costituenti il comando in ASCII normale. Quando il file finisce il SuperScript mette uno $00, l'EasyScript no.

Il GeoWrite utilizza un formato tutto particolare. Cominciamo dalla directory. Ecco un esempio di file GeoWrite, così, come compare in directory (i codici sono in esadecimale):

```
83 01 11 --nome-- 01 09 01 07 56 04
07 0F 04 10 00
```

Il primo $83 è quello che dice al DOS che si tratta di un file USR. I successivi due byte ($01 e $11) sono rispettivamente traccia e settore di un blocco che noi chiameremo «pointer block». Dopo il nome, scritto QUASI normalmente (come dirò dopo), i primi due byte ($01 e $09) sono traccia e settore di un blocco che chiameremo «identify block». Dopo $01 e $07 (che non so neanch'io cosa sono, ma se li toccate guai!) segue la data, nella forma anno/giorno/mese (avete fatto caso che $56 = 86 dec?). $0F e $04 rappresentano l'ora, in questo caso le 15:04, anzi le 3:04 PM, all'americana! Gli ultimi due byte ($10 e $00) sono come al solito la lunghezza in blocchi, nel formato lo/hi.

Cominciamo allora a vedere cos'è quello che abbiamo chiamato identify block. Si tratta di un blocco che definisce univocamente un file GeoWrite. Contiene, al suo interno, i dati per lo sprite che compare sullo schermo (presumibilmente), le scritte «Write Image v1.1» e «Geo Write v1.1», e le «info» sul programma. Se analizzate un file GeoPaint, vi accorgerete che porta scritto «Paint Image v1.1» e «Geo Paint v1.1». Capito il trucco?

La struttura tipica di un pointer block è la seguente:

```
00 FF 01 04 00 FF 00 FF 00 FF ...
```

Dopo i primi due ($00 e $FF), ogni coppia di byte indica traccia e settore del primo blocco di file per ogni pagina. Nell'esempio riportato, la prima pagina comincia nel blocco 1;4, e non ci sono altre pagine. Se ad esempio avessimo avuto:

```
00 FF 01 04 02 01 02 12 00 FF ...
```

il file sarebbe costituito da 3 pagine, che cominciano rispettivamente a 1;4, 2;1 e 2;18 (18 decimale = $12).

Per trovare il file bisogna dunque districarsi tra questi rimandi, un po' come in una caccia al tesoro. In effetti, a partire dal blocco 1;4, troveremo il nostro bel file scritto come un file sequenziale, dove cioè i primi 2 byte indicano il blocco successivo. Uno $00 in prima posizione significa che si tratta dell'ultimo blocco della pagina ed il byte successivo, anziché indicare un settore, indica i byte occupati in questo blocco. Se ad esempio abbiamo $00 $27, significa che nel blocco sono occupati $27 (=39) byte, ed il 39esimo byte contiene $00, se il file finisce lì, o $1C se si tratta di un salto di pagina.

Le particolarità del file non finiscono qui. Il file vero e proprio, vale a dire ciò che abbiamo scritto, è preceduto da 24 byte, che probabilmente indicano variabili tipo margine sinistro, destro, font, eccetera. Dalla posizione $1A del primo blocco del file iniziano i codici corrispondenti al file che abbiamo scritto. Anche questi codici, tanto per esser coerenti, sono particolari: nel normale codice ASCII (quello usato dal SuperScript, per intenderci), le minuscole hanno codici compresi tra $41 e $5A, e le maiuscole tra $C1 e $DA. Nei file GeoWrite, invece, le minuscole vanno da $61 a $7A, le maiuscole da $41 a $5A. Vengono trattate così tutte le lettere che il Geos considera, anche i nomi nella directory (per questo in modo maiuscolo/grafica si vedono solo tanti segni, anziché i nomi dei file). Tutti gli altri codici (numeri e segni di interpunzione) sono «normali».

Abbiamo così finito di esaminare, seppure in modo molto sommario, le caratteristiche dei file su cui dobbiamo lavorare. Vediamo allora come lavora Geo Convert.

Alla linea 160 il programma sposta i puntatori del Basic, «chiudendogli» una parte della memoria, che verrà utilizzata come buffer per il file che deve essere convertito. Alle linee 170-200 si inizializza il linguaggio macchina: viene caricata la seconda parte del programma, in LM, ed impostate le variabili per la chiamata delle subroutine in LM. Alle linee 210-400 si ha la presentazione del programma e la richiesta dei nomi del file. Trucchetto: per ef-

```
100 rem"
110 rem"   *   Geo Convert v1.2
120 rem"
130 rem"   di Daniele  Finocchiaro
140 rem"         gennaio 1987
150 rem"
160 poke 56,76:poke 55,0:clr
170 if peek(49538)<>204 then load "geo convert.lm",8,1
180 poke 49196,24:poke 49197,76
190 l1=49152:l2=49209
200 l3=49485:l4=49517
210 poke 53280,11:poke 53281,12:            print "(blk)(clr)"chr$(14)chr$(8)
220 print "┌──────────┬──────────────────────────────┐";
230 print"├──────────┤ (wht)Geo Convert v1.2(blk) ├──────────┤";
240 print"├──────────┴──────────────────────────────┤";
250 print"│ (cyn) di (yel)Dany Finocchiaro(cyn) (gennaio 1987)(blk)  │";
260 print"└─────────────────────────────────────────┘";
270 print"  programma di conversione files testo"
280 print"da SuperScript (o EasyScript) a GeoWrite":print:print
290 :
300 open 1,0
310 print "(blk)File (yel)SuperScript (blk)[::::::::::::::::::](wht)(up)"
320 print "(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rg
ht)(rght)(rght)(rght)(rght)(rght)(rght)(rght)";
330 input#1,n1$:n1$=left$(n1$,16):print
340 if n1$="" then run
350 print "(blk)File (yel)GeoWrite     (blk)[::::::::::::::::::](wht)(up)"
360 print "(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rg
ht)(rght)(rght)(rght)(rght)(rght)(rght)(rght)";
370 input#1,n2$:n2$=left$(n2$,16)
380 if n2$="" then run
390 if len(n2$)<16 then n2$=n2$+chr$(160):goto 390
400 print : print "(down)(down)": poke 198,0
410 print "(up)(gry3)          INSERISCI DISCO SORGENTE
420 for i=1 to 90:next i
430 print "(up)(blk)          INSERISCI DISCO SORGENTE
440 for i=1 to 87:next i
450 if peek(198)=0 goto 410
460 get a$:print "(up)
470 :
480 open 15,8,15
490 open 5,8,5,"0:"+n1$+",s,r"
500 input#15,a,a$,b,c:if a=0 goto 550
510 print "(up)(blk)Errore:(wht)"a"(left), "a$","b"(left),"c
520 print "(blk)Premi un tasto":poke 198,0:wait 198,1:get a$
530 close 5:close 15:run
540 :
550 sys 11 :p1=19456:p2=peek(49196)+peek(49197)*256:poke p2,0
560 close 5:close 15:sys 65511
570 nb=int((p2-p1)/254)-((p2-p1)/254<>int((p2-p1)/254))+2
580 if nb<29 goto 610
590 print "(wht)ATTENZIONE:(blk) Il file e' troppo lungo"
600 goto 520
610 for i=p1 to p1+23:read a:poke i,a:next
620 data 0,0,223,1,40,0,223,1,223,1,223,1,223,1,223,1,223,1,223,1,23,9,0,0
630 poke 198,0
640 print "(up)(wht)          INSERISCI DISCO GEOS
650 for i=1 to 90:next i
660 print "(up)(blk)          INSERISCI DISCO GEOS
670 for i=1 to 87:next i
680 if peek(198)=0 goto 640
690 get a$:print "(up)
700 :
710 open 15,8,15
720 open 3,8,3,"#"
730 gosub 1100:pt=t:ps=s
740 gosub 1100:it=t:is=s
750 print#15,"b-p"3;0
760 sys 12
770 print#15,"u2:"3;0;it;is
780 print#15,"s0:"+n2$:open 2,8,2,"0:"+n2$+",s,w"
790 poke 173,p1/256:poke 172,p1-peek(173)*256
800 poke 175,p2/256:poke 174,p2-peek(175)*256
810 sys 13
820 input#15,a,a$,b,c:if a=0 then 870
830 print "(blk)Errore:(wht)"a"(left),"a$","b"(left),"c
840 print "(blk)Premi un tasto":poke 198,0:wait 198,1:get a$
850 close 3:close 15:run
860 :
870 t=18:s=1
880 print#15,"u1:"3;0;t;s
890 get#3,a$:t1=asc(a$+chr$(0)):get#3,a$:s1=asc(a$+chr$(0))
900 a9=0:for i=5 to 229 step 32
910 print#15,"b-p:"3;i
920 a$="":for j=1 to 16:get#3,a1$:a$=a$+a1$:next
930 if a$<>n2$ goto 950
940 a9=i-3:i=254
950 next i
960 if a9=0 then t=t1:s=s1:goto 880
970 print#15,"b-p:"3;a9+1:get#3,a$,a1$:ft=asc(a$):fs=asc(a1$+chr$(0))
980 print#15,"b-p:"3;a9
990 c$=chr$(131)+chr$(pt)+chr$(ps)+left$(n2$,16)+chr$(it)+chr$(is)+chr$(1)
1000 c$=c$+chr$(7)+chr$(87)+chr$(1)+chr$(1)+chr$(17)+chr$(0)+chr$(nb)+chr$(0)
1010 print#3,c$;
1020 print#15,"u2:"3;0;t;s
1030 print#15,"b-p"3;0
1040 print#3,chr$(0)chr$(255)chr$(ft)chr$(fs);
1050 sys 14
1060 print#15,"u2:"3;0;pt;ps
1070 close 3:print#15,"i":close 15:end
1080 :
1090 :
1100 rem  subroutine che alloca il
1110 rem  primo blocco libero
1120 :
1130 t=19:s=0
1140 print#15,"b-a:"0;t;s
1150 input#15,a,a$,b,c
1160 if a<>65 then 1190
1170 if b=18 then t=17:s=0:goto 1140
1180 t=b:s=c:goto 1140
1190 if a=0 then return
1200 print"(wht)Errore:(blk)"a"(left), "a$","b"(left),"c
1210 print"(wht)Premi un tasto":poke 198,0:wait 198,1:get a$
1220 close 3:close 15:run
```

fettuare un input senza il punto interrogativo, basta dare un OPEN 1,0 e poi prendere l'input da questo file (INPUT#1,...). Ulteriori vantaggi: la presenza di due punti o virgole a destra del cursore non causano un messaggio? Extra ignored; il computer non accetta il return se non si è immesso nulla (vantaggio che io non ho sfruttato).

Nelle linee 480-620 avviene la lettura del file SuperScript (o EasyScript, dato che il formato è lo stesso). A parte alcuni «convenevoli» in Basic, la lettura vera e propria avviene con una routine in LM, allocata a $C000-$C039. Questa routine legge un byte alla volta dal bus seriale; se si tratta di un $80, lo ignora. Qualsiasi altro carattere viene memorizzato e se è una lettera viene convertita come abbiamo detto sopra. Evidentemente anche i comandi SuperScript verrano copiati lasciando a discrezione dell'utente la loro cancellazione tramite GeoWrite.

Arrivati a questo punto il programma richiede l'inserimento del disco Geos dove va memorizzato il file convertito. Fatto ciò, inizia a memorizzare i vari blocchi.

Nota

I codici di controllo nei listati sono riportati in forma «esplicita», in conseguenza dell'impiego della stampante Star NL-10 e relativa interfaccia per Commodore. Ovviamente, nella digitazione del programma é necessario usare i consueti tasti che corrispondono alle indicazioni fra parentesi: ad esempio cursore destro per (RGHT), CTRL-3 per (RED) eccetera.

(CLR)	-	(YEL)	-
(HOME)	-	(RVS)	-
(DOWN)	-	(OFF)	-
(UP)	-	(ORNG)	-
(RGHT)	-	(BRN)	-
(LEFT)	-	(LRED)	-
(BLK)	-	(GRY1)	-
(WHT)	-	(GRY2)	-
(RED)	-	(LGRN)	-
(CYN)	-	(LBLU)	-
(PUR)	-	(GRY3)	-
(GRN)	-	(SWLC)	-
(BLU)	-		

Alle linee 730-740 vengono allocati due blocchi, per l'identify e per il pointer block. Si provvede subito (linee 750-770 e LM da $C039 a $C14C) a scrivere l'identify block i cui byte sono memorizzati, assieme al linguaggio macchina, alle locazioni $C04C-$C14.

Per scrivere il file ho utilizzato un trucchetto: il file viene scritto come sequenziale, per aumentare la velocità di scrittura, dopodiché si cambiano alcuni byte nella directory per «aggiustare» il tutto e far diventare il file USR. Ecco quindi (linee 780-850 e LM $C14D-$C16C) la scrittura del file SEQ. Nota: la routine in LM è molto simile a quella del Kernal SAVE e ne utilizza alcune routine ($EDDD, $FCDB, $FCD1, etc.). Per chi volesse saperne di più consiglio «Il S.O. del CBM 64», edito dalla EVM.

Dalla linea 870 in poi si gestisce la directory. Viene cercata la posizione dove è stato scritto il file SEQ appena creato (linee 870-960). Quindi si leggono traccia e settore del primo blocco del file (linea 970), e vengono messi nelle variabili FT e FS.

Alle linee 980-1020 avviene la riscrittura della directory nel formato ora visto. Infine (linee 1030-1060 e LM $C16D-$C184) si scrive il pointer block.

Ultima nota sulla subroutine 1100. Essa serve ad allocare il primo blocco libero che il DOS si trova sotto mano e viene usata per sapere dove andare a mettere il pointer e l'identify block.

Acris in fundo: il file da convertire dev'essere più corto di 29 blocchi circa, altrimenti non entra in una sola pagina ed il GeoWrite segnala «Page too large»: provare per credere.

Sarebbe d'altronde troppo difficile gestire da programma il salto pagina, perché la lunghezza di questa varia col variare dei font e degli stili utilizzati. A questo proposito, vorrei far notare che, convertendo un file abbastanza lungo, non lo si vedrà comparire tutto sullo schermo, dopo averlo caricato in GeoWrite. Niente paura: il testo c'è ma non entra nella pagina fisica gestita dal GeoWrite. Basta, per farlo ricomparire, mettere un salto di pagina vicino alla fine di questa (Options-Page break) oppure diminuire la grandezza dei caratteri che si vedono. Il testo verrà «tirato fuori» dalla linea di fine pagina.

Le istruzioni finiscono qui. Il programma in sé è di utilizzo immediato, ma sapere come lavora un programma Basic è sempre meglio. Comunque, spero che i più «smanettoni» del 64 prendano l'occasione offerta dal Geos al volo e scrivano, che so, una routine di conversione Geo Paint-Koala, o un Icon Switcher (alla MacIntosh...). Per ora, tanti saluti.

Geo Couvert L.M.

```
100 rem"
110 rem"       LM DATA
120 rem"
130 rem" questo programma crea un file
140 rem" contentente il linguaggio
150 rem" macchina necessario a
160 rem" Geo Convert v1.2
170 rem"
180 print "(clr)"
190 for i=49152 to 49539
200 read a:poke i,a:ck=ck+a
210 next i
220 if ck<>41466 then print "errore nei data!":end
230 poke 251,peek(45):poke 252,peek(46)
240 poke 43,0:poke 44,192:poke 45,132:poke 46,193
250 save "geo convert.lm",8
260 poke 43,1:poke 44,8:poke 45,peek(251):poke 46,peek(252):print "ok!":end
265 :
270 data 162, 5, 32, 198, 255, 32, 183, 255, 41, 64, 240
280 data 1, 96, 32, 207, 255, 201, 128, 240, 241, 201, 64
290 data 48, 19, 201, 91, 16, 5, 105, 32, 76, 43, 192
300 data 201, 192, 48, 6, 201, 219, 16, 2, 233, 128, 141
310 data 24, 76, 238, 44, 192, 208, 210, 238, 45, 192, 76
320 data 5, 192, 162, 3, 32, 201, 255, 160, 0, 185, 76
330 data 192, 32, 221, 237, 200, 208, 247, 76, 204, 255, 0
340 data 255, 3, 21, 191, 255, 255, 255, 128, 0, 1, 143
350 data 255, 1, 136, 1, 1, 139, 255, 193, 138, 0, 65
360 data 138, 255, 241, 138, 128, 17, 138, 142, 17, 138, 128
370 data 17, 138, 191, 145, 138, 128, 17, 138, 159, 17, 138
380 data 128, 17, 138, 191, 145, 142, 128, 17, 130, 191, 145
390 data 131, 128, 17, 128, 128, 17, 128, 255, 241, 255, 255
400 data 255, 131, 7, 1, 0, 0, 255, 255, 0, 0, 87
410 data 114, 105, 116, 101, 32, 73, 109, 97, 103, 101, 32
420 data 86, 49, 46, 49, 0, 0, 0, 0, 0, 0, 0
430 data 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
440 data 0, 0, 0, 0, 0, 0, 103, 101, 111, 87, 114
450 data 105, 116, 101, 32, 32, 32, 32, 86, 49, 46, 49
460 data 0, 0, 0, 0, 162, 16, 189, 124, 40, 157, 107
470 data 40, 202, 208, 247, 32, 75, 193, 169, 0, 133, 46
480 data 32, 60, 57, 169, 40, 0, 3, 169, 108, 133, 2
490 data 32, 17, 57, 165, 5, 201, 0, 208, 4, 165, 4
500 data 201, 96, 144, 8, 169, 0, 133, 5, 169, 96, 133
510 data 4, 169, 96, 56, 229, 4, 133, 24, 169, 0, 229
520 data 5, 133, 25, 70, 24, 169, 0, 133, 25, 24, 169
530 data 210, 101, 24, 133, 24, 144, 2, 230, 25, 169, 10
540 data 133, 5, 169, 1, 133, 56, 169, 57, 133, 55, 169
550 data 32, 32, 69, 193, 169, 40, 133, 3, 169, 108, 133
560 data 2, 32, 72, 193, 169, 32, 32, 69, 193, 169, 1
570 data 133, 56, 0, 162, 2, 32, 201, 255, 160, 0, 177
580 data 172, 32, 221, 237, 32, 219, 252, 32, 209, 252, 144
590 data 243, 32, 254, 237, 169, 2, 32, 195, 255, 76, 204
600 data 255, 0, 162, 3, 32, 201, 255, 160, 126, 169, 0
610 data 32, 221, 237, 169, 255, 32, 221, 237, 136, 208, 243
620 data 76, 204, 255
```

File utilizzati

300: OPEN 1,0 apre un canale dalla tastiera, da cui si preleverà l'input.

480,710: OPEN 15,8,15 apre un canale di comunicazione col disco

490: OPEN 5,8,5 apre il file SEQ del SuperScript in lettura

720: OPEN 3,8,3,«#» apre un file random

Ricordo, a proposito della sub 1100, che se si tenta di allocare un blocco già occupato, il DOS dà un errore 65 (No block) e comunica traccia e settore del primo blocco libero in BAM

Locazioni di memoria particolari:

160: poke 56,76: poke 55,0 sposta il top della memoria usata dal Basic da 40960 (valore standard) a 19456 (=76*256), in modo da usare l'area 19456-40960 come buffer per il file letto. Quando si smette di usare il programma, è consigliabile dare un reset (sys 64760 o sys 64738), per ripristinare tutte le locazioni ai valori standard

210: 53280 e 53281 contengono i codici di colore dello schermo e del bordo

400,630,etc: POKE 198,0 azzera il buffer di tastiera

Tutti gli altri valori utilizzati (49152, 19456, etc.) servono al funzionamento delle sub in LM o per memorizzare il file.

Variabili principali:

L1: indirizzo di start della routine LM di lettura

L2: indirizzo di start della routine LM per scrivere l'identify block

L3: indirizzo di start della routine LM per scrivere il file SEQ (che in seguito viene trasformato in USR)

L4: indirizzo di start della routine LM per scrivere il pointer block

N1$: nome del file SuperScript

N2$: nome del file GeoWrite

P1: indirizzo del primo byte del file in memoria

P2: indirizzo dell'ulimo byte del file in memoria

NB: numero dei blocchi occupati dal file GeoWrite

PT,PS: traccia e settore del pointer block

IT,IS: traccia e settore dell'identify block

AS: posizione, all'interno del blocco, dove inizia la descrizione del file SEQ, che subito dopo viene trasformato in USR

Lettura

```
., c000 a2 05     ldx #$05
., c002 20 c6 ff  jsr $ffc6
., c005 20 b7 ff  jsr $ffb7
., c008 29 40     and #$40
., c00a f0 01     beq $c00d
., c00c 60        rts
., c00d 20 cf ff  jsr $ffcf
., c010 c9 80     cmp #$80
., c012 f0 f1     beq $c005
., c014 c9 40     cmp #$40
., c016 30 13     bmi $c02b
., c018 c9 5b     cmp #$5b
., c01a 10 05     bpl $c021
., c01c 69 20     adc #$20
., c01e 4c 2b c0  jmp $c02b
., c021 c9 c0     cmp #$c0
., c023 30 06     bmi $c02b
., c025 c9 db     cmp #$db
., c027 10 02     bpl $c02b
., c029 e9 80     sbc #$80
., c02b 8d 18 4c  sta $4c18
., c02e ee 2c c0  inc $c02c
., c031 d0 d2     bne $c005
., c033 ee 2d c0  inc $c02d
., c036 4c 05 c0  jmp $c005
```

Scrittura dell'identify block

```
., c039 a2 03     ldx #$03
., c03b 20 c9 ff  jsr $ffc9
., c03e a0 00     ldy #$00
., c040 b9 4c c0  lda $c04c,y
., c043 20 dd ed  jsr $eddd
., c046 c8        iny
., c047 d0 f7     bne $c040
., c049 4c cc ff  jmp $ffcc
```

Scrittura file

```
., c14d a2 02     ldx #$02
., c14f 20 c9 ff  jsr $ffc9
., c152 a0 00     ldy #$00
., c154 b1 ac     lda ($ac),y
., c156 20 dd ed  jsr $eddd
., c159 20 db fc  jsr $fcdb
., c15c 20 d1 fc  jsr $fcd1
., c15f 90 f3     bcc $c154
., c161 20 fe ed  jsr $edfe
., c164 a9 02     lda #$02
., c166 20 c3 ff  jsr $ffc3
., c169 4c cc ff  jmp $ffcc
```

Scrittura pointer block

```
., c16c 00        brk
., c16d a2 03     ldx #$03
., c16f 20 c9 ff  jsr $ffc9
., c172 a0 7e     ldy #$7e
., c174 a9 00     lda #$00
., c176 20 dd ed  jsr $eddd
., c179 a9 ff     lda #$ff
., c17b 20 dd ed  jsr $eddd
., c17e 88        dey
., c17f d0 f3     bne $c174
., c181 4c cc ff  jmp $ffcc
```

C$: sequenza di byte che viene scritta nella directory

FT,FS: traccia e settore del primo blocco del file vero e proprio.

Comandi DOS utilizzati

PRINT#15,«B-A:»0;17;1 alloca sulla BAM il blocco 17;1 del disco montato sul drive 0. Se il blocco è già occupato, il DOS segnala un errore 65 e comunica il primo blocco libero del disco

PRINT#15,«B-P:»3;0 sposta il buffer pointer all'inizio (posizione 0) del buffer riservato al file random 3. Se si vogliono scrivere o leggere byte a partire da una posizione specifica (come avviene alle linee 970 e 980) basta spostare il buffer pointer alla posizione voluta

PRINT#15,«U2:»3;0;17;1 scrive sul disco, nel blocco 17;1 del drive 0, il contenuto del buffer corrispondente al file n. 3

PRINT#15,«U1:»3;0;17;1 legge e deposita nel buffer corrispondente al file n. 3, il contenuto del blocco 17;1

Linguaggio macchina

La routine $C000-$C039 legge e converte il file SuperScript. Innanzitutto definisce il canale # 5 come input ($FFC6=CHKIN del Kernal), poi controlla che il file non sia finito ($FFB7=READST ritorna nell'accumulatore il registro di stato, se è uguale a $40 il file è finito). Nel caso il file non sia finito, si legge un byte ($FFCF=CHRIN) e lo si converte come dovuto. I byte vengono memorizzati a partire da $4C18, ed il puntatore ($C02C-C02D) viene incrementato dallo stesso programma ($C02E-$C033).

La routine $C039-$C04C trasferisce al disco le locazioni di memoria tra $C04C e $C14B, che contengono i byte che vanno scritti nell'identify block. Routine del Kernal utilizzate: $FFC9 (CHKOUT, apre un canale di output) e $EDDD (trasferisce un byte alla porta seriale).

La routine $C14D-$C16C scrive sul disco tutti i byte letti dalla routine di lettura ($C000). $AC e $AD contengono l'indirizzo di partenza del file in memoria, nel formato lo/hi (scritti da basic alla linea 790), e $AE-$AF quello di fine file (scritti alla linea 800). La routine è molto simile a quella del Kernal SAVE ($F5DD), e utilizza, come quella, alcune routine molto utili del Kernal: $FCDB incrementa il contenuto di $AC/$AD, $FCD1 lo confronta con $AE/$AF. La routine $FCD1 ritorna al Carry impostato a 1 se i due valori ($AC/$AD e $AE/$AF) sono uguali. $EDFE e $FFC3 sono usate anche dal Kernal per chiudere il file.

La routine $C16D-$C184 manda una sequenza di $00 e $FF al disco, per 126 volte (126 volte * 2 byte a volta + 4 byte scritti da Basic = 256 byte che riempiono il pointer block)

Identify block

```
.:c04c 00 ff 03 15 bf ff ff ff
.:c054 80 00 01 8f ff 01 88 01
.:c05c 01 8b ff c1 8a 00 41 8a
.:c064 ff f1 8a 80 11 8a 8e 11
.:c06c 8a 80 11 8a bf 91 8a 80
.:c074 11 8a 9f 11 8a 80 11 8a
.:c07c bf 91 8e 80 11 82 bf 91
.:c084 83 80 11 80 80 11 80 ff
.:c08c f1 ff ff ff 83 07 01 00
.:c094 00 ff ff 00 00 57 72 69
.:c09c 74 65 20 49 6d 61 67 65
.:c0a4 20 56 31 2e 31 00 00 00
.:c0ac 00 00 00 00 00 00 00 00
.:c0b4 00 00 00 00 00 00 00 00
.:c0bc 00 00 00 00 00 67 65 6f
.:c0c4 57 72 69 74 65 20 20 20
.:c0cc 20 56 31 2e 31 00 00 00
.:c0d4 00 a2 10 bd 7c 28 9d 6b
.:c0dc 28 ca d0 f7 20 4b c1 a9
.:c0e4 00 85 2e 20 3c 39 a9 28
.:c0ec 00 03 a9 6c 85 02 20 11
.:c0f4 39 a5 05 c9 00 d0 04 a5
.:c0fc 04 c9 60 90 08 a9 00 85
.:c104 05 a9 60 85 04 a9 60 38
.:c10c e5 04 85 18 a9 00 e5 05
.:c114 85 19 46 18 a9 00 85 19
.:c11c 18 a9 d2 65 18 85 18 90
.:c124 02 e6 19 a9 0a 85 05 a9
.:c12c 01 85 38 a9 39 85 37 a9
.:c134 20 20 45 c1 a9 28 85 03
.:c13c a9 6c 85 02 20 48 c1 a9
.:c144 20 20 45 c1 a9 01 85 38
.
```