CommunITel Random Access Procedures

INTRODUCTION

The CommunITel Viewdata System uses a proprietary random access filing system for storing Viewdata pages (frames). This filing system overcomes the 31 file limit of DFS on the BBC microcomputer. The frames are stored within a single large file named VWDB. This file occupies almost the whole of the disc surface and can hold 95 frames on a 40 track disc or 195 frames on an 80 track disc. The structure of the VWDB file is shown below.

OVERALL STRUCTURE

Byte	Contents
0 - 26	Catalogue heading
27 - 511	Gap - unused
&100 - &10FF	Catalogue of Frames 256 available entry slots, numbered 0 - 255 Each entry slot is 16 bytes in size.
&1100 - EOF	Frame Table Frames stored sequentially Each frame is 1024 (&400) bytes in size Space for 195 frames on 80T disc or 95 frames on 40T disc

The Catalogue of Frames and the Frame Table comprise the random access element of the system. Frame IDs are hashed and the resulting hash number is used as a pointer into the Catalogue of Frames. This pointer is used to extract an offset into the Frame Table and this offset allows the frame to be directly loaded and displayed on screen.

Following is a set of procedures and functions which can be used to maintain the database of frames. These functions and procedures are early prototypes. They do not allow for the gap between the 27th byte and the 511th. This gap was introduced at a later date.

The hashing algorithm is written in machine code for speed but is not especially efficient and results in many collisions. For example frame IDs 0a, 1a, 2a, …, 8a and 9a all hash to the same number. To overcome these collisions the hash number is incremented until a free location is found, if writing a frame, or until the frame ID and the search ID match. The hash number is modulo 256 and can wrap round to 0 if needed.

There is space in the Catalogue of Frames for 256 entries, of which, as a maximum, 195 are used. This allows between 25% and 60% free space, which should allow rapid location of the requested frame.

The procedures can be found in the file 'IT' on the SSD image.

The 'IT' code listing follows.

```
 10*K.0 PROCOPEN("HELP")|M
   20*K.1 PROCCAT|M
   30*K.2 PROCSAVE(&7C00,"
   40*K.3 PROCLOAD(&7C00,"
   50*K.4 PROCDELETE("
   60*K.5 PROCCREATE("
   70*K.7 """)|M"
   80*K.8 *DUMP HELP|M
   90*K.9 CLOSE#0|M
  100:
  110ONERROR:ON ERROR OFF:REPORT:PRINT" AT LINE
";ERL'"PTR=";PTR#A%'"EXT=";EXT#A%':CLOSE#0:END
  120PROCINIT
  130END
  140:
  150DEFPROCINIT
  160DIM GPBLOCK 16
  170PROCCRCASS
  180errnum=42
  190PROCASSFOLTS
  200ENDPROC
  210:
  220DEFPROCCREATE(N$)
  230LOCAL A%,T$,D$
  240A%=OPENUPN$
  250CLOSE#A%
  260IF A% PRINT"FILE EXISTS, DO YOU WISH TO OVERWRITE?"; : IF NOT
FNGETYN ENDPROC
  270A%=OPENOUTN$
  280BPUT#A%,&AA : BPUT#A%,&55 : BPUT#A%,&AA : BPUT#A%,&55
  290INPUT"TITLE",T$
  300PROCSPUT(T$,9)
  310REM FAKE CRC
  320BPUT#A%,0 : BPUT#A%,0
  330D$="X X X"
  340PROCSPUT(D$,8)
  350REM ACCESS
  360BPUT#A%,0
  370REM NR FILES IN USE
  380BPUT#A%,0
  390REM NR FILES DELETED
  400BPUT#A%,0
  410INPUT"MAX NUMBER OF FRAMES",D%
  420REM MAX NR FILES
  430BPUT#A%,D%
  440FOR Z%=0 TO 255
  450PROCPUTREC(0,"",0,0)
  460NEXT
  470PTR#A%=PTR#A%+D%*1024
  480CLOSE#A%
  490ENDPROC
  500:
```

These key definitions were used during the development of the suite of functions and procedures to test the random access filing and database manipulation.

The error trapping is set up to allow easy access to information about the random access pointers

This is the initialisation section to set up the global variable, open the random access file and assemble the hashing and error reporting machine code.

All further operations are manual by means of the function keys.

PROCCREATE is the main procedure used to create a new random access database file. This is a prototype structure and does not entirely match the final, released version, which includes the gap between bytes 27 to 511 inclusive.

```
  510DEFPROCPUTFOUR(F%)
  520LOCAL Z%
  530!GPBLOCK=F%
  540FOR Z%=0 TO 3
  550BPUT#A%,GPBLOCK?Z%
  560NEXT
  570ENDPROC
  580:
  590DEFPROCSPUT(A$,MAX)
  600LOCAL Z%,B$
  610REM PUT THE STRING UP AND PAD TO LENGTH
  620FOR Z%=1 TO MAX
  630B$=LEFT$(A$,1)
  640IF B$<>"" BPUT#A%,ASCB$ : ELSE : BPUT#A%,32
  650A$=RIGHT$(A$,(LENA$-1))
  660NEXT
  670ENDPROC
  680:
  690DEFFNGETYN
  700LOCAL G%
  710REPEAT
  720G%=GET
  730UNTIL G%=ASC"Y" OR G%=ASC"y" OR G%=ASC"N" OR G%=ASC"n"
  740VDUG%,10,13
  750=G%=ASC"Y" OR G%=ASC"y"
  760:
  770DEFPROCSAVE(WHERE%,FR$)
  780IF FNSEARCH(FR$) GOTO810
  790IF MAXNR%=FRINU% CALL full
  800IF MAXNR%=FRINU%+FRDEL% PROCGETDEL
  810PROCSAVEDAT(WHERE%,FR$)
  820ENDPROC
  830:
  840DEFPROCLOAD(WHERE%,FR$)
  850IF FNSEARCH(FR$) PROCLOADDAT(WHERE%) ELSE CALLnofile
  860ENDPROC
  870:
  880DEFPROCDELETE(FR$)
  890IF FNSEARCH(FR$) AND INUSE%=&80: PTR#A%=CATENT%: FRINU%=FRINU%-1
: FRDEL%=FRDEL%+1 :PROCPUTREC(&40,FRAME$,oFSET%,LODADD%):PROCPUTNRS:
ELSE: CALLnofile
  900ENDPROC
  910:
```

This will write 4 bytes to the Catalogue of Frames for the load addresses of the particular frame. These are not used in the released version.

The procedure writes the frame ID as a 10 character string, end padded with spaces, into the Catalogue of Frames.

A simple function to return whether yes or no (Y or N) was entered.

Writes the frame into the Frame Table and updates the Catalogue of Frames. Over-writes a deleted frame but will ignore a currently used frame

Loads the requested frame ID to the specified location.

The delete procedure only changes the usage status from used to deleted. Used = &80. Deleted =&40. Unused = &00

```
  920DEFFNSEARCH(FR$)
  930FR$=FNSTRIP(FR$)
  940LOCALGOT%,PRESC%,T$,START%
  950PROCGETCATHEAD
  960PTR#A%=FNCRC(FR$)*16+&1B
  970START%=PTR#A%
  980REPEAT
  990CATENT%=PTR#A%
 1000IF PTR#A%>255*15+&1A PTR#A=&1B : PRINT"LOOOOOOOOPY":VDU7
 1010GOT%=BGET#A%
 1020IF GOT%=0 : PRESC%=FALSE : UNTIL1 : GOTO1060
 1030T$=FNSGET(10)
 1040IF T$=FR$ PRESC%=TRUE : UNTIL1 : GOTO1060
 1050PTR#A%=PTR#A%+5 : IF PTR#A%=START% PRINT"CATALOGUE FULL" : STOP :
ELSE : UNTIL0
 1060INUSE%=GOT%:FRAME$=T$:oFSET%=BGET#A%:LODADD%=FNGETFOUR
 1070=PRESC%
 1080:
 1090DEFPROCGETCATHEAD
 1100LOCAL Z%,T%
 1110PTR#A%=4
 1120TITLE$=FNSGET(9)
 1130PTR#A%=PTR#A%+2 : REM SKIP CRC
 1140DATE$=FNSGET(8)
 1150ACCESS=BGET#A%
 1160IF ACCESS=0 ACCESS$="open" ELSEACCESS$="closed"
 1170FRINU%=BGET#A% : REM GET FRAMES IN USE
 1180FRDEL%=BGET#A% : REM GET FRAMES DELETED
 1190MAXNR%=BGET#A% : REM GET MAX FRAMES ALLOWED
 1200ENDPROC
 1210:
 1220DEFPROCCRCASS:LOCAL Z,u,e,b,a,d,q,c:DIMCRC 55:c=&80:DIM string 10

1230FORZ=0TO1:P%=CRC:[OPT0:ldy#0:styc+5:styc+4:.u:ldy#0:.q:lda(c),Y:ld
x#8:.b:lsrA:rolc+4:rolc+5:bcca:pha:ldac+4:eor#&2D:stac+4:pla:.a:dex:bn
eb:incc:bned:incc+1:.d:ldac:cmpc+2:bneq:ldac+1:cmpc+3:bneq:rts:]NEXT:E
NDPROC
 1240:
 1250DEFFNCRC($string):LOCAL c:c=&80:!c=string:c!2=string+LEN
$string:CALLCRC:=(c!4AND&FF)
 1260:
 1270DEFFNSGET(MAX%):LOCAL
ST%,Z%,TEMP$:ST%=PTR#A%+MAX%:REPEATZ%=BGET#A%:IF
A%=32:UNTIL1:PTR#A%=ST%:=FNSTRIP(TEMP$)
 1280TEMP$=TEMP$+CHR$Z%:UNTILPTR#A%=ST%:=FNSTRIP(TEMP$)
 1290:
 1300DEFFNGETFOUR
 1310LOCALZ%
 1320FOR Z%=0 TO 3
 1330Z%?string=BGET#A%
 1340NEXT
 1350=!string
 1360:
```

The search function returns TRUE if the frame ID is found. Strips spaces from the ID and reads the catalogue header.

FNCRC is the hashing algorithm which returns a pointer into the Frame Catalogue. Adds one to pointer until frame ID found

Reads the catalogue header information.

CRC, date and Access Byte are not used in the released version.

Assembles the machine code hashing routine. They called it a CRC

FNCRC passes the frame ID to the hashing routine and returns the hash number

FNSGET returns a stripped string from the database

FNGETFOUR returns a 4 byte address from the database. Not used by the released version.

```
1370DEFPROCSAVEDAT(PL%,A$)
1380IF INUSE%=&20 FRINU%=FRINU%+1 : FRDEL%=FRDEL%-1
1390IF INUSE%=0 oFSET%=FRINU%+FRDEL% : FRINU%=FRINU%+1
1400IF INUSE%=&40 FRDEL%=FRDEL%-1 : FRINU%=FRINU%+1
1410INUSE%=&80
1420LODADD%=PL%
1430PTR#A%=CATENT%
1440PROCPUTREC(INUSE%,A$,oFSET%,LODADD%)
1450PROCPUTNRS
1460PROCPUTK(oFSET%,LODADD%)
1470ENDPROC
1480:
1490DEFPROCPUTREC(B%,C$,D%,E%)
1500BPUT#A%,B%
1510PROCSPUT(C$,10)
1520BPUT#A%,D%
1530PROCPUTFOUR(E%)
1540ENDPROC
1550:
1560DEFPROCPUTNRS
1570PTR#A%=&18
1580BPUT#A%,FRINU%
1590BPUT#A%,FRDEL%
1600ENDPROC
1610:
1620DEFPROCCAT
1630PROCGETCATHEAD
1640PRINTTITLE$ TAB(20)"Current date "DATE$
1650PRINT"Subdirectory: "SUBDIR$ TAB(20)"Access: "ACCESS$
1660PRINT"Inuse: ";FRINU% TAB(20)"Unused: "MAXNR%-FRINU%
1670PRINT
1680IF FRINU%=0 ENDPROC
1690FOR Z%=1 TO FRINU%
1700REPEAT
1710GOT%=BGET#A%
1720IF GOT% <>&80: PTR#A%=PTR#A%+15 : UNTIL0
1730UNTIL1
1740PRINTTAB((Z%-1)MOD4*10)FNSGET(10)" ";
1750PTR#A%=PTR#A%+5
1760NEXT
1770PRINT
1780ENDPROC
1790:
1800DEFPROCOPEN(N$)
1810CLOSE#0
1820A%=OPENUPN$
1830SUBDIR$=N$
1840FOR Z%=0 TO 1
1850T%=BGET#A% : IF T%<>&AA CLOSE#A%:CALLnotvalid
1860T%=BGET#A% : IF T%<>&55 CLOSE#A%:CALLnotvalid
1870NEXT
1880ENDPROC
1890:
```

Writes the frame into the Frame Table and updates the Catalogue of Frames. Over-writes a deleted frame but will ignore a currently used frame

Writes the entry into the Catalogue of Frames.

Updates the catalogue header with the number of frames in use and the number of deleted frames

Displays a catalogue of the saved frames including the database header information. Frame IDs are in 'as found' order.

Opens a viewdatabase and checks for a valid file. A valid file has &AA, &55, &AA and &55 at the start.

Removes leading and trailing spaces from a string passed to the function and then returns it.

```
 1900DEFFNSTRIP(A$)
 1910REPEAT
 1920IF LEFT$(A$,1)=" " A$=RIGHT$(A$,LENA$-1) : UNTIL0
 1930UNTIL1
 1940REPEAT
 1950IF RIGHT$(A$,1)=" " A$=LEFT$(A$,LENA$-1) : UNTIL0
 1960UNTIL1
 1970=A$
 1980:
```

Writes the frame data into the Frame Table. The start address is the prototype address. The final address starts at &1100 not &1B. Uses OSGBPB.

```
 1990DEFPROCPUTK(O%,P%)
 2000O%=(O%+4)*1024+&1B
 2010PTR#A%=O%:FOR z%=0 TO 1023:BPUT#A%,P%?z%:NEXT
 2020ENDPROC
 2030?GPBLOCK=A%:LOCAL
A%:GPBLOCK!1=P%:GPBLOCK!5=1024:GPBLOCK!9=O%:A%=1:X%=GPBLOCK MOD
256:Y%=GPBLOCK DIV 256:CALL &FFD1:ENDPROC
 2040DEFPROCGETK(O%,P%)
 2050O%=(O%+4)*1024+&1B
 2060PTR#A%=O%:FOR z%=0 TO 1023:P%?z%=BGET#A%:NEXT:ENDPROC
 2070?GPBLOCK=A%:LOCAL
A%:GPBLOCK!1=P%:GPBLOCK!5=1024:GPBLOCK!9=O%:A%=3:X%=GPBLOCK MOD
256:Y%=GPBLOCK DIV 256:CALL &FFD1:ENDPROC
```

PROCGETK reads a frame from the Frame Table. The start address is the prototype address. The final address starts at &1100 not &1B. Uses OSGBPB.

```
 2080DEFPROCLOADDAT(W%)
 2090PROCGETK(oFSET%,W%)
 2100ENDPROC
 2110DEFPROCTEST
 2120FOR TEST=0 TO 20
 2130CLS
 2140$&7D00=STR$TEST
 2150PROCSAVE(&7C00,STR$TEST)
 2160NEXT
 2170ENDPROC
 2180DEFPROCLEST
 2190FOR TEST=0 TO 170
 2200CLS
 2210PROCLOAD(&7C00,STR$TEST)
 2220IF $&7D00<>STR$TEST VDU7
 2230NEXT
 2240ENDPROC
```

Test procedures for the prototype database. These are not part of the commercial release.

```
2250DEFPROCASSFOLTS
2260DIM FOLTSTORE 80
2270P%=FOLTSTORE
2280[OPT2
2290.notvalid
2300brk
2310]
2320?P%=errnum
2330$(P%+1)="not a vaild subdirectory"
2340P%=P%+(LEN$(P%+1)+2)
2350P%?-1=0
2360[OPT2
2370.full
2380brk
2390]
2400?P%=errnum+1
2410$(P%+1)="subdirectory full"
2420P%=P%+(LEN$(P%+1)+2)
2430P%?-1=0
2440[OPT2
2450.nofile
2460brk
2470]
2480?P%=errnum+2
2490$(P%+1)="no such file in subdirectory"
2500P%=P%+(LEN$(P%+1)+2)
2510P%?-1=0
2520ENDPROC
2530DEFPROCTIT
2540FOLTSTORE=&6000
2550errnum=42
2560PROCASSFOLTS
2570ENDPROC
2580DEFPROCGETDEL
2590PTR#A%=&1B
2600REPEAT
2610CAPENT%=PTR#A%
2620GOT%=BGET#A% : PTR#A%=PTR#A%+15
2630UNTILGOT%=&40
2640PTR#A%=CAPENT%+11
2650oFSET%=BGET#A%
2660PTR#A%=CAPENT%
2670PROCPUTREC(0,"",0,0)
2680INUSE%=&20
2690ENDPROC
```

Assembles a set of machine code error message routines.

The following routines are also just for testing the prototype.