A A R D V A R K

DATA SHEET

SUBJECT: How to use BASIC Command Files on 65D Disk Systems

INTRODUCTION

If you have had to key in the same set of BASIC commands repeatedly, you will appreciate knowing how to use BASIC Command files. By definition, a BASIC Command file is a file or list of BASIC commands that has been stored for "deferred" execution. Imagine being able to switch your console keyboard to a deferred execution mode to allow entry of a list of commands for execution at a later time. That, in effect, describes the capability inherent in a BASIC Command file.

A better description of the capability of a command file is: deferred console input list. That is because it is not limited to just BASIC commands. The file may contain any valid input response to what the computer expects at the time each file entry is "read" by the computer. When the BASIC Monitor is in the "COMMAND" mode, it expects either BASIC commands, line numbered BASIC statements or "immediate execution" BASIC statements. When it is in the "RUN" mode, it expects a response to any "INPUT <variable list>" type program statement. The latter requires pre-determination of the logic path to be followed by each program invoked by the command file.

By now, it should be apparent that realizing the BASIC command file capability that is inherent in all OSI Operating Systems adds another control and utility dimension to your computer system. And, the command file usage is not just limited to BASIC. It can be used just as well with the ASSEMBLER, or any other use, because all uses expect control commands to be entered via the console input device, which does not have to be the keyboard.

GENERAL PROCEDURE

The basic steps necessary to invoke a BASIC command file are as follows:

1. Build and store a command file on a disk data file.
2. Transfer the command file from disk to the indirect file and/or "memory device" file.
3. Establish the command file as the "console input device".

It is not absolutely necessary to transfer the command file from disk to memory, but it is better for overall general use. To use a disk file as the "console input device" requires using device# 6 or 7, which may also be used by a program invoked by the command file. The same problem exists, to a lessor extent, for memory device# 5, which also may be used by a program invoked by the command file. However, the "memory input" file pointer locations can be saved and then restored by each using program, precluding possible interference with their use by a command file. Also mutually exclusive blocks of memory can be reserved for use by programs versus command files to further preclude interference. In short, use of memory as the "console input device" for command files is a better choice.

BUILDING A COMMAND FILE ON DISK

The AARDVARK ASCII FILE/TEXT EDITOR is ideally suited for creating and maintaining (modifying) command files on disk. For those who have it, just invoke function #1 - 'ENTER NEW FILE FROM KEYBOARD" - enter the command file name and unit and then enter your "command lines". For those who do not have the ASCII EDITOR (get it), use the simple program shown in Appendix A. Just key in the program and follow its instructions.

To get you started, the following is a sample command file that you can use for testing purposes.

```
DISK!"LO BEXEC*
LIST
RUN"PAUSE":REM SEE APPENDIX B
DISK!"LOAD DIR"
LIST
```

The "PAUSE" program shown in appendix B does just that: it provides a pause in the execution of the command file so you can see what has happened.

TRANSFER OF THE COMMAND FILE TO MEMORY

Appendix C contains a listing of a program that allows transfer of ASCII files between disk and memory. It is specifically designed to transfer command files to memory for easy execution. The program sets the indirect file and the memory device# 5 input/output pointers equal to the address of the first byte above the end of the current work space. The address of the last byte in the work space, in decimal, is PEEK(132)+PEEK(133)*256. The program assumes that PEEK(8960) contains the page number address of the last page of RAM physically available in your system. If this location has been modified by running "CHANGE" or by some other means, then "this" program will not know what the memory capacity of your system is. It is best to just use locations 132 and 133 to "tell" the operating system the address of the end of your work space; and the best way to establish that is in "BEXEC*", as opposed to "CHANGE". Whatever method is used on your system, the program in Appendix C will only achieve satisfactory results when there is sufficient memory above the work space to hold the command file, as indicated to the program by the contents of locations 8960, 132 and 133.

ESTABLISH THE COMMAND FILE AS THE CONSOLE INPUT DEVICE

Assuming you have used the Appendix C program successfully to transfer a command file to memory, then all you have to do to execute it is invoke either of the following commands:

COMMAND	FUNCTION
<CTRL-X>	Makes the indirect file located at PEEK(9368)*256, the console input device.
DISK!"IO 10"	Makes memory device# 5, located at PEEK(9098)+PEEK(9099)*256, the console input device.

Actually the console input device, in both cases, is memory device# 5, but the implementation is not exactly the same.

Since the "input pointers" for both the indirect file and memory device# 5 are initialized to the same address, either of the above commands will accomplish the same thing. Using the <CTRL-X> command is better because it takes less key strokes and because the command file can be executed repeatedly. Both methods use the auto incrementing input file pointers at locations 9098 and 9099. Each time a <CTRL-X> is invoked, the effect is a

```
POKE9098,0:POKE9099,PEEK(9368):DISK!"IO 10"
```

That is why the <CTRL-X> command will allow you repeated executions of the command file. The DISK!"IO 10" command tranfers control input to the "current" position of the memory device# 5 file, as indicated by the current contents of locations 9098 and 9099.

RETURN OF CONTROL BACK TO CONSOLE KEYBOARD

When you run your first command file, you will notice a "SN ERROR" message at the end of your run followed by a return of control to the console keyboard. That is because an invalid BASIC command, namely ".END", was used as an "end of file" marker. When the BASIC Monitor encounters an invalid command, or an error of any type, it returns control to the permanent console input device, which is PEEK(10950).

The explicit way of returning or transferring control to the permanent, or other, console input device is to use the DISK!"IO xx" command. Whenever the BASIC monitor encounters this command, it unconditionally transfers control to the device indicated by the "xx", whether it exists or not. So, you could use a DISK!"IO 01" or a DISK!"IO 02" command to restore control to the serial device or polled keyboard respectively. This may be "neater", but it requires unneccessary housekeeping.

This does imply further flexibility; you can transfer control out of your "main" command file/device to another and then back again; i.e., if you can retain the pointer position in your "main" command file so that you can continue from where you were when you exited. This is relatively easy to accomplish if your "main" command file is attached to device# 5. This does set a little complicated, but it is another indication of the great flexibility "hidden" in the OSI Operating Systems.

CLOSING

Many examples could be provided to illustrate applications of command files, but that is a broad enough subject to be covered in a separate data sheet - depending on the response to this one. In the meantime, your imagination will provide the limit of what you can use command files for. OSI's Operating Systems leaves the door wide open. Enjoy.

APPENDIX A

```
10 REM PROGRAM TO COPY FROM KB TO DISK
11 REM
15 POKE 2888,0:POKE2972,13:POKE2976,13
16 REM
20 REM THIS PROGRAM NEEDS 1 DISK BUFFER
21 REM SO RUN"CHANGE" TO ASSIGN ONE
22 REM BEFORE KEYING IN AND "SAVING" IT
23 REM
30 INPUT"ENTER FILE NAME";F$
40 INPUT"ENTER UNIT";U$
50 DISK!"SE "+U$:DISK OPEN,6,F$
60 PRINT:PRINT"ENTER DATA LINES. TERMINATE WITH '.END'"
70 PRINT
100 INPUTA$:PRINT#6,A$:IFLEFT$(A$,4)<>".END"GOTO100
200 DISK CLOSE,6
300 DISK!"SE A
400 POKE 2888,0:POKE2972,58:POKE2976,44
500 END
```

APPENDIX B

PAUSE PROGRAM

```
10 X=PEEK(10950)
20 PRINT"PAUSE AND LOOK, HIT RETURN TO CONTINUE";
30 INPUT#X,A$
40 END
```

APPENDIX C

```
10 MI=9098:MO=9105:LP=8960:EWS=132
15 TT$=CHR$(93):D$=CHR$(34)
20 TOP=(PEEK(8960)+1)*256:LWSA=PEEK(EWS)+PEEK(EWS+1)*256
25 AVAIL=TOP-LWSA-1:IF AV<=0THENPRINT"NO SPACE ABOVE WS AVAIL":END
30 MEM=INT(LWSA/256)+1
35 POKE 2888,0:POKE 8722,0:POKE 2972,13:POKE 2976,13
40 FORI=1TO24:PRINT:NEXTI
50 PRINTAV;"BYTES ABOVE THE WORK SPACE IS AVAILABLE.  OK";
55 INPUTA$
60 A$=LEFT$(A$,1):IFA$=""ORA$="Y"GOTO 70
65 GOTO440
70 POKE9368,MEM:POKE9554,MEM
99 PRINT
100 PRINT"MEMORY <--> DISK COPY PROGRAM"
110 PRINT:PRINT"OPTIONS ARE:"
120 PRINT"     1. COPY FROM DISK TO MEMORY"
130 PRINT"     2. COPY FROM MEMORY TO DISK"
139 PRINT:PRINT
140 INPUT" ENTER OPTION #,ELSE NULL TO ESCAPE";OPT
145 ON OPT+1 GOTO 440
150 ON OPT GOTO 200,200: GOTO110
200 IF OPT=1 THEN T$="INPUT":RF=6:WF=5
210 IF OPT=2 THEN T$="OUTPUT":RF=5:WF=6
220 PRINT"ENTER "T$" FILENAME";:INPUT F$
230 INPUT"ON WHICH UNIT";U$:DISK!"SE "+U$
240 DISK OPEN,6,F$
260 POKE MO,0:POKE MO+1,MEM
270 POKE MI,0:POKE MI+1,MEM
300 REM
310 INPUT #RF,A$
330 REM
340 LC$="":IF LEFT$(A$,1)=" " THEN LC$=G$
350 WB$=LC$+A$
355 IF LEFT$(WB$,4)=".END" OR LEFT$(WB$,1)=TT$ GOTO 400
360 PRINT #WF,WB$
375 PRINT WB$
380 GOTO 310
390 REM
400 PRINT#WF,".END":PRINT".END"
425 DISK CLOSE,6
440 POKE 2888,27:POKE 8722,27:POKE 2972,58:POKE 2976,44
450 DISK!"SE A"
460 END
```