Support Group Application Note
Number: 034
Issue: 1
Author:

Acorn

EdBin and Move User Guide

Applicable
Hardware :
BBC Master 512

Related
Application
Notes:

Support Group
Acorn Computers Limited
Acorn House
Vision Park
Histon
Cambridge CB4 4AE

1.1 Introduction

EdBin allows you to EDit BINary files, ie executable object code files or data files that cannot be loaded into ordinary word processor or text editor. The file can be examined after loading into memory and modified if necessary. Commands are available to edit the file using hexadecimal or ASCII values, initialise blocks of the file to a specified constant, copy blocks to different locations, search for a specific string and compare two blocks of the file. After editing, the data can be written back to the original file or to a different file if required. All EdBin commands can be terminated at any time using CTRL-C or suspended using CTRL-S. If the output is suspended using CTRL-S it can be restarted using any key except CTRL-C, CTRL-S or BREAK. The EdBin program can be found on the DOS Plus Boot Disc, disc 1.

1.2 Starting EdBin

To start EdBin type:

    EDBIN <filename>

where <filename> is the file that you want to edit. The file will be loaded into memory by EdBin when it starts up and it will then respond with a ' - ' prompt to indicate that it is ready to accept commands.

Alternatively, EdBin can be started up by simply typing:

    EDBIN

EdBin will be loaded and respond with the prompt as before but no file will be loaded to edit. In this mode memory can be examined and modified and written to a disc file. If at any time the user wants to edit a file rather than memory the file can be loaded in using the R(EAD) <filename> command which loads the file in as if EDBIN <filename> had been typed.

1.3 EdBin Commands

All the EdBin commands are single letter commands followed by up to three parameters. The following table summarised the available commands.

Command	**Function**	
C <range> <address>	C(ompare)	blocks of memory
D (<range>)	D(ump)	memory in hex and ASCII formats
E <address>	E(dit)	memory using hex or ASCII values
F <range> <byte | word>	F(ill)	memory with a constant value
H	H(elp)	give summary of commands
M <range> <address>	M(ove)	blocks of memory
Q	Q(uit)	return to DOS + command prompt
R <filename>	R(ead)	file into memory for editing
S <range> <"string">	S(earch)	memory for specified string
W (<filename>)	W(rite)	memory to disc file

where: <address> = (<segment>:) <offset>
<range> = (<segment>:) <start offset> <end offset>

The above commands are now explained in more detail. Where <offset> is used it refers to a hexadecimal offset address which can be entered as 1 to 4 digits -leading zeros can be omitted ie 7A can be entered as 7A, 07A or 007A. If more than 4 hex digits are entered the most significant digits will be truncated ie 12345 will be treated as 2345.

Where an <end offset> is required it refers to the end offset + 1, ie,

D 100 180

will dump the bytes between 100H and 17FH inclusive. to specify the end address of a 64k segment in 16 bits an end offset of 0 must be used ie,

F C000 0 E5

will fill the last 4 Kbytes of the current segment with the byte E5H.

Where <segment> is used it refers to a 80186 segment address which can also be entered as 1 to 4 hex digits but must be followed immediately by a : to indicate that it is a segment value. In all relevant commands below if no segment address is specified that most recently specified value is used or the segment address of the file buffer if no previous value has been specified. For all commands any leading or trailing spaces will ignored. Items enclosed in <> brackets indicate parameters that the command must have, items also enclosed in () brackets indicate optional parameters that do not have to be specified. Items separated by the | character indicate that the command can take several different parameters but only one can be specified at a time.

All commands can be entered in upper or lower case (or both).

C(ompare)

Syntax: C (<segment>:) <source start> <source end> (<segment>:) <dest start>
where <start> = <start offset> ; <end> = <end offset>
<dest> = <destination>

Function: Compare the specified block of memory with another block of the same size given by the destination start address.

Description: If there are any differences found between the two specified blocks the are reported as follows:

<source address> <source byte> <dest address> <dest byte>

The address is displayed in standard segment:offset form and the byte at that address is displayed in both hexadecimal and ASCII forms.

If there are no differences between the two blocks EdBin will not display anything and simply return with the ' - ' prompt to indicate that the blocks are identical.

The Compare command can be used on blocks up to 64Kbytes in length by specifying an end offset of 0 for the source end address. If a compare of greater than 64Kbytes is required it can be done in blocks of 64Kbytes.

Example: C 300 500 1000
will compare 200H bytes at 300H to 4FFH in the current segment with 200H bytes and 1000H bot 11FFH in the same segment.

C 0 0 2000:0
compares the whole 64Kbytes of the current segment with another 64K block at 2000:0

D(ump)

Syntax: D (<segment>:) (<start offset>) (<end offset>)

Function: Display the specified area of memory.

Description: The memory contents are displayed as hex values and ASCII characters. Each line of the memory dump displays the memory address in segment:offset form followed by 16 bytes o hex values and 16 bytes of ASCII characters. Characters outside the ASCII range 20H-7EH are shown as a full stop. All the parameters in the above command are optional. If the segment address is omitted the last value will be used, if the start and end offsets are omitted the last end address is used as the start address and the last end address + 080H is used as the end address, if the end address is omitted the start address + 080H is used.

Note - the above applies only if no other intervening commands have been used which change the current segment address or current offset address.

The minimum number of bytes that can be displayed is 16 bytes ie

D 100 105

will display 100H to 10FH.

Example: D 1000 1580

will display memory from 1000H to 157FH inclusive in the current segment in the format described above. If the dump command is subsequently used with no parameters, ie,

D

128 bytes of memory from 1580H to 15FFH will be displayed. Further D commands entered with no parameters will display the next 128 bytes of memory.

E(dit)

Syntax: E (<segment>:) <start offset>

Function: Allows memory contents to be examined and new valued to be entered if required.

Description: A line of 16 bytes of memory is displayed in hex and ASCII formats initially with the cursor under the least significant digit of the first byte specified. Cursor movement and data entry is controlled using the following keys:

LEFT	move cursor left, if at far left display previous 16 bytes
RIGHT	move cursor right, if at far right display next 16 bytes
UP	display next 16 bytes
DOWN	display previous 16 bytes
SHIFT-LEFT	move cursor to far left of current field
SHIFT-RIGHT	move cursor to far right of current field
COPY-DELETE	toggle between hex field and ASCII field

The display consists of two 16 byte fields - a hex display and an ASCII display. The COPY-DELETE keys are used to switch between the two fields. (NOTE - on the Master 512 COPY is used as the ALTernate shift key of an IBM keyboard hence it should not be used like a SHIFT key ie to toggle between the data entry fields COPY should be pressed down first and held while DELETE is pressed). While the cursor is in th hex field data is entered in hex digits, each digit being shifted in from the right. To advance to the next field the normal cursor keys are used. SHIFTed cursor keys are used to move to the far left or right of the current field. If the cursor is in the ASCII field data is entered as ASCII bytes. The cursor is automatically advanced to the next field to allow text to be typed in directly. When text is entered at the far right of the field the next 16 bytes are automatically displayed to allow typing to continue over 16 byte boundaries.

NOTE - if data is being entered in the ASCII field, control codes in the range 0 to 31 can be entered by pressing the appropriate key in conjunction with the CTRL key. ie CTRL-Z will enter the end of text market 1AH. The only exception is CTRL-C (value 03H) which is used to terminate the command. If the user wishes to enter this code it must be entered in the hex data field as hex digits 03.

The E(dit) command is terminated by pressing CTRL-C.

<u>Example:</u> E 3000:560
will display
3000:560 41 20 62 6F 74 20 6F 66 20 74 65 78 1A 07 00 A bot of text....

The spelling mistake in the text can be corrected by two methods:

1. Move the cursor to the fourth byte in the hex field and enter the hex digits 6 and 9.
2. Use COPY-DELETE to move to the ASCII field of the display and move to the fourth byte in this field and enter the character i.

CTRL-C is then entered to terminate the command for both methods.

The editing can be checked by for example entering the command D 560 570 which should display
3000:560 41 20 62 69 74 20 6F 66 20 74 65 78 74 1A 07 00 A bit of text.....

F(ill)

<u>Syntax:</u> F (<segment>:) <start offset> <end offset> <fill byte | word>

<u>Function:</u> Fills memory with a constant value between the specified addresses.

<u>Description:</u> The constant used can be specified as a byte or word value. If a word value is specified the least significant byte is written first. An end offset of 0 can be used to specify a fill operation to the last byte in the specified segment.

<u>Example:</u> F 1000 1010 55
will fill bytes 1000H to 100FH inclusive with the value 55H.

F 1000 1010 1234
will fill bytes 1000H to 100FH inclusive with the word 1234H with the least significant byte written first.

H(elp)

<u>Syntax:</u> H

<u>Function:</u> Give help information.

Description: The H command displays the EdBin version number and a list of available commands with their required syntax. It does not take any parameters.

Example: H

M(ove)

Syntax: M (<segment>:) <source start> <source end> (<segment>:) <dest start>
where <start> = <start offset> ; <end> = <end offset>
<dest> = <destination>

Function: Move a block of memory to specified address.

Description: The specified block of memory is moved to the destination address. The block of memory at the source address is not generally affected by the move - it is simply copied to the given address. if an overlapping move is specified (ie a move where part of the destination block overlaps the source block) the data that could be overwritten by the operation is always moved first to ensure that the destination data is correct. The user should be aware that in these cases the source block must inevitably be corrupted in the overlapping area.

Example: M 100 740 1200:10
moves 640H bytes from 100H to 73FH to address 10H in segment 1200H.

Q(uit)

Syntax: Q

Function: Exit EdBin and return to the DOS + command line prompt.

Description: The Q command is used with no parameters. It closes the currently active file (if one exists), restores the DOS+ system environment and returns to the DOS+ command line prompt.

Example: Q

R(ead)

Syntax: R <filename>

Function: Read a file into memory.

Description: The specified file is read into memory at the start of the file buffer. The size of the file and the number of bytes actually read are displayed as the file is being loaded into memory. These two numbers should normally be the same but in the event of a disc error the number of bytes read will indicate how much of the file can be recovered.

When a file is loaded using the Read command the specified <filename> becomes the currently active file (CAF). The will remain active until a subsequent Read operation which will close the previous CAF and the newly loaded file becomes the new CAF.

NOTE - if EdBin is executed with a filename on the command line ie,

EDBIN TEST.DAT

then TEST.DAT will automatically be loaded into memory as if a R TEST.DAT command had been entered. The file TEST.DAT therefore becomes the currently active file. (For more information of the use of the currently active file see the Write command).

Example: R FILE.EXE

loads the file FILE.EXE into memory at the start of the file buffer and reset the current command segment/offset variables so that file editing commands will operate on the file buffer.

NOTE - EdBin will always load the file into memory at the start of its file buffer regardless of the type of file ie CMD or EXE files will not have the file header stripped off - the file will be loaded into memory in the same form as it appears on disc. The user should therefore be aware that in the above two cases the start of the code segment will be offset by the length of the header.

S(earch)

Syntax: S (<segment>:) <start offset> <end offset> <"string">

Function: Search memory fo a specified text string.

Description: If any matching text strings are found it is reported as the start address of the string in segment:offset form. The address given is of the first byte of the matching string. The search string must be enclosed in double quotes (") and can be up to 72 characters long. (Maximum length for complete command line is 80 characters). The end offset specified is the end address +1 of the search area so to allow the search to continue right up to the end of a segment an end address of 0 can be specified ie

S 4000 0 "eric"

will search from 04000H up to 0FFFFH inclusive. The condition for a string to be found is that it must be completely contained within the search area, ie if string "eric" lives at 03FFDH then

S 0 4000 "eric"

will not report it but if our string "eric" lives at 03FFCH then the above search will find it. Any 8 bit character string can be searched for using escape sequences to allow control codes and characters above 07FH to be specified. (NB these are compatible with the MOS escape sequences). The | character is used to denote an escape sequence.

The following table shows how all the characters are specified.

<table>
<tr><th>String</th><th>Hex Byte</th></tr>
<tr><td>"|@"</td><td>0</td></tr>
<tr><td></td><td></td></tr>
<tr><td>"|a" or "|A"</td><td>1</td></tr>
<tr><td>to</td><td>to</td></tr>
<tr><td>"|z" or "|Z"</td><td>1A</td></tr>
<tr><td></td><td></td></tr>
<tr><td>"|["</td><td>1B</td></tr>
<tr><td>to</td><td>to</td></tr>
<tr><td>"|_"</td><td>1F</td></tr>
</table>

String   Hex Byte
" "   20
to   to
"~"   7E

except for following two special cases:
""""   22
"||"   7C

Delete is entered as follows:
"|?"   7F

Characters above 128 can be entered using the |! operator ie:
"|!<char>"   80-FF
where <char> is any of above 7 bit chars

Note - the characters in the search string are case dependent ie "ABC" will match "ABC" only not "abc" or "AbC" etc.

Any escape arguments not recognised are reduced to the argument alone ie "|1" is reduced to "1" etc. and any surplus redundant "|!" operators are ignored ie "|!|!|@" is reduced to "|!|@".

Any string not terminated by a " character or containing an odd number of " characters will be reported as bad string ie:

String   Error
"abc   No terminating "
"ab""   Single quote character in string
"ab"""c"   as above

A Bad String error will also be generated if no argument is supplied for the escape character | or if a null string or equivalent is specified ie

String   Error
"a|"   No escape argument
""   Null string
"|!"   Reduced to null string hence as above

The search operation can be terminated at any time by pressing CTRL-C.

Example: S 100 7B00 "a text string"
will search from 100H to 7AFFH for the string "a text string". Any occurrences of the string in the specified area might be reported as:

01F3:0151
01F3:0279
01F3:7A01

W(rite)

Syntax: W (<filename>)

Function: Writes the specified file or currently active file to a disc file.

Description: If the W command is used with no <filename> the modified data in memory is written back to the currently active file that was specified in the last R command. If a <filename> is specified the data is written instead to this specified file. Note that using the W command with a filename does not affect the currently active filename - therefore subsequent W commands with no filename will write the data to the filename specified in the last R command.

When writing out the file the number of bytes written is displayed on the screen - if EdBin cannot write out all the bytes to disc an error message will be given to indicate that the disc is probably full.

Example: W
writes data from the file buffer to the currently active filename. The number of bytes written is equal to the length of the file loaded in by the R command.

1.4 Error Messages

The following table lists the errors that may occur while using EdBin:

Error Message	Cause
File not found	The R command has been used to try and read in a file that does not exist on the disc or the full pathname for the file has not been specified.
Path not found	The R <filename> or W <filename> command has been used to read or write a file in a directory that does not exist.
Too many open files	The R command or W <filename> command has been used to try to open a file when there are already 20 open files. This error should nor normally occur and it will only happen if some previous application has not closed its files on exit. EdBin requires only 2 files open at any time.
Access denied	The R command has been used on a Read Only file. The R command attempts to open a file for R/W access assuming that the user may want to write the modified data back to the same file. To prevent this error the file should be set to R/W using FSET before entering EdBin. This error may also occur if W <filename> has been used to write to a file that has the same name as an existing file that is marked R/O or if the specified directory is full.
Illegal file handle	The W command has been used when there is no Currently Active File to write the data to. The CAF must be activated by an R command or by specifying a filename in the command line when EdBin is started up.

MOVE Utility

Syntax:
MOVE <source filespec> <-fs> [/option] <dest dirspec> <-fs> [/option]

Explanation:
The MOVE command copies files between DOS+ and BBC filing systems. It is intended primarily for saving DOS+ files on ANFS or ADFS but some support is provided for other filing systems (ie DFS). In this document the DOS+ operating system will be referred to as DOS and BBC filing systems (ANFS and ADFS) will be referred to as MOS filing systems. Files that have been originally save under DOS+ ie utilities such as DISK.CMD, COLOUR.EXE or text files such as LETTER.DOC are referred to as DOS PLUS files and file that have been originally saved under ANFS or ADFS ie BBC BASIC files or VIEW documents are referred to as MOS files. The MOVE utility enables DOS PLUS files to be transferred from DOS to MOS and then copied back from MOS to DOS with the original filename, filetype and attributes preserved. MOS files can be copied to DOS but the filename by be truncated and file attributes will not be saved.

The <source filespec> can be used to specify a single file or multiple files using the wild card characters * and ? for DOS and * and # for MOS. The source files can be specified using any valid DOS or MOS pathname. MOVE does not recognise passwords in DOS file specifications - if files are password protected a default password must be set.

The <dest dirspec> is used to identify the destination directory which can be specified using any valid DOS or MOS pathname. The special character @ is used to indicate the currently selected destination directory for both DOS and MOS operations. (NOTE - as @ is a valid filename character under DOS is should not be used as a directory name as a conflict will occur). The /r option can be used in a destination specification to allow single files to be copied to the given filename rather than being copied with the same name to the given directory.

The <-fs> field is used to indicate the source or destination filing system as follows:

Filing System	<-fs>
DOS PLUS	-dos
ADFS	-adfs
ANFS	-net
DFS	-disc

The [/option] field specifies the following optional parameters:

Option	Description
/C	Only valid for MOS source filespec - allows filetype subdirectories to be copied from MOS to MOS (Default for MOS to MOS copy is to ignore FSD's)
/|	Only valid fro MOS source filespec - specifies that filetype subdirectories should be ignored. It is only needed in MOS to DOS copies as it is the default setting for MOS to MOS copies. It allows DOs files with no extension to be coped from MOS.
/S	Only valid for source files. It is used to copy DOS system files.
/R	Only valid for destination files and single file copies. It is used to specify a destination filename instead of a directory name.

Example:
A>MOVE SDIR.CMD -DOS :1.$.DOSPLUS -ADFS

This copies the file SDIR.CMD from the current directory on DOs drive A to the directory :1.$.DOSPLUS on ADFS drive 1.

Copying files between DOS and MOS

This section explains how to copy a group of files from DOs to a MOS filing system (ANFS in this example) and then how to copy them back to DOS. It is assumed that the user has already logged on to the network by some means (ie the STAR utility), has created a directory called DOSPLUS in the Users Root Directory (URD), and it is the Currently Selected Directory (CSD). (See ANFS documentation for explanation of URD, CSD). It is not absolutely necessary to create a separate directory for DOs files but it is STRONGLY RECOMMENDED as mixing DOS and MOS files in the same directory will probably cause confusion and may cause MOS files to be copied unnecessarily during MOS to DOS transfers.

The following command will copy all the files from DOs drive B with the BAT extension to the CSD on ANFS (ie DOSPLUS) (NOTE - the A> prompt should not be typed in the following examples - it is included to indicate the current DOS drive)

A>MOVE B:*.BAT -DOS @ -NET

To copy the files back from ANFs to the CSD on DOS drive A type

A>MOVE BAT.* -NET @ -DOS

NOTE that the source specification for the ANFS files reverses the filetype and filename of the DOS file specification. This is necessary because of the way that DOs files are stored under ANFS or ADFS (See later section).

To copy all the files from subdirectory GEM on DOS drive A to the same directory on ANFS use

A>MOVE \GEM\ *.* -DOS @ -NET

To copy all the files back from ANFS to DOS directory GEMBAK on drive B type

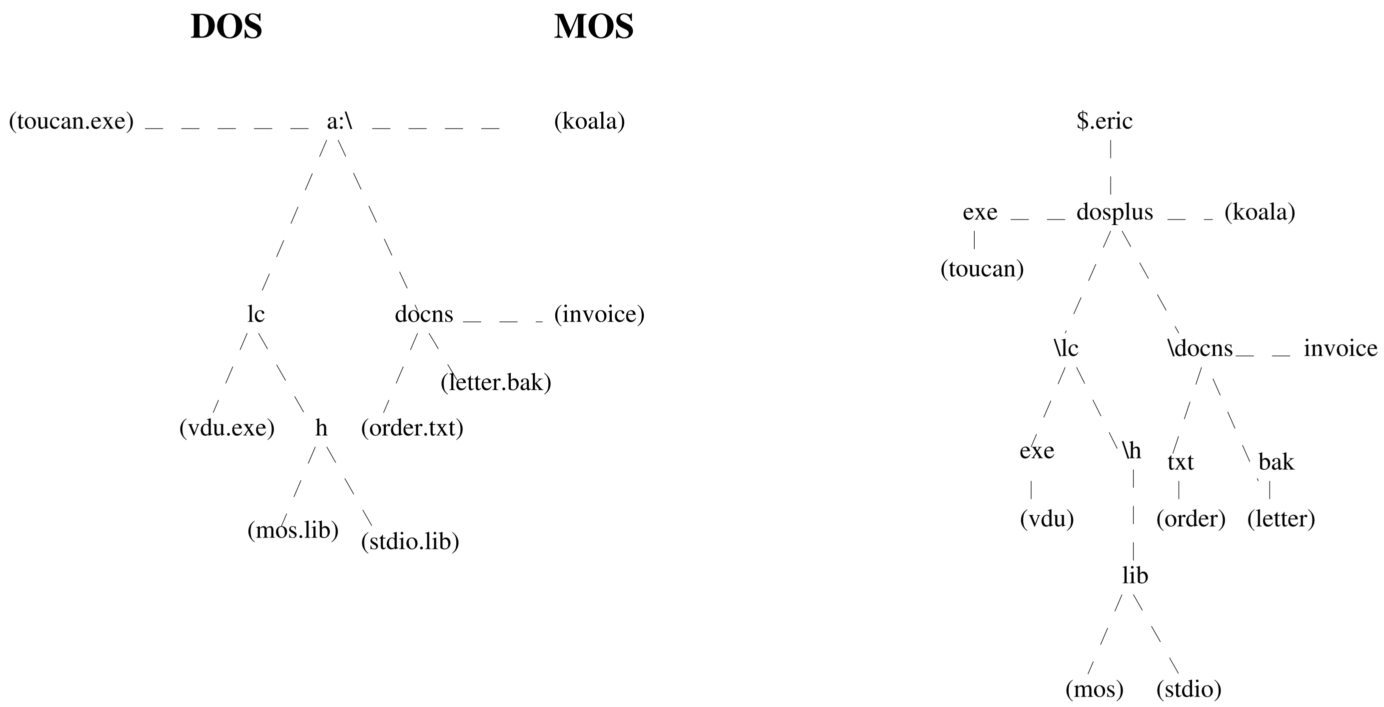
A>MOVE * -NET B:\GEMBAK -DOS

NOTE that for the ANFS source specification * is equivalent to the DOS *.* and copies all the files from the current directory.

Copying directories between DOS and MOS

The above example shows how to copy files from DOS to a single MOS directory ie all the files were being copied to the same ANFS directory. This method is adequate if the source DOS disc does not contain any DOS subdirectories (DSD's) or if only a small number of files are to be copied. However if the source DOS disc contains several subdirectories (which may themselves contain further subdirectories) then copying all the files to the same ANFS directory would result in loss of the DOS directory structure. To allow DSD's to be created in an MOS directory the special character \ is used as the first character of the DSD name. This is necessary because of the way that DOS files are stored in MOS filing systems. DOS files that have no filetype extension ie INVOICE are stored in the specified destination directory. Files that have a filetype extension ie LETTER.BAK are stored in a directory BAK with the name LETTER. The filetype subdirectory (FSD) is automatically created if not already present. Similarly the file VDU.EXE would be stored in an FSD called EXE with the name VDU. As DOS filetype extensions can be no longer than 3 characters any subdirectories with a name longer than 3 characters will be ignored in a MOS to DOS file copy. This provides another means of distinguishing between DSD's and FSD's in the same MOS directory ie if the DSD name is longer than 3 characters it will not be mistaken for a filetype subdirectory. The following diagram illustrates the mapping between a DOS directory structure and an equivalent MOS directory structure.

DOS To MOS Directory Mapping



Where object enclosed in () are files, objects not enclosed in () are directories

The above example shows how files would be copied from DOs drive A: to an ANFS directory $.eric.dosplus ie to directory DOSPLUS in user ERIC on the network. It is important to note that the DOS Root Directory (DRD) DOSPLUS and DOS subdirectories \LC, \DOCNS etc are not automatically created by the MOVE utility ie MOVE does not copy a whole directory tree. It is the users responsibility to create the DRD's and DSD's when they are required. This is directly analogous to using the DOS PLUS COPY command which copies files between directories but requires the destination directory to be already present.

The character \ is used above to indicate DOS subdirectories in a MOS directory rather than use names of 4 or more characters. Either method could be used interchangeably but it is better to just choose the one method that suits you and use it consistently.

The actual commands required to perform the above MOVE operation are now given as an example. It is assumed that the following MOS * commands have already been given to prepare for the copy and ERIC is already a user of the NET.

```
*net
*i am eric                  log on to net
*cdir dosplus               create DOS Root Directory
*cdir dosplus.\lc           create DOS SubDirectories
*cdir dosplus.\lc.\h
*cdir dosplus.\docns
```

<u>Copy all files in root on DOS drive A:</u>

A>MOVE A:*.* -DOS DOSPLUS -NET

<u>Copy all files in A:\DOCNS</u>

A>MOVE A:\DOCNS*.* -DOS DOSPLUS.\DOCNS -NET

<u>Copy all files in A:\LC</u>

A>MOVE A:\LC*.* -DOS DOSPLUS.\LC -NET

<u>Copy all files with type LIB in A:\LC\H</u>

A>MOVE A:\LC\H*.LIB -DOS DOSPLUS.\LC.\H -NET

The commands required for the inverse operation (ie copying from the NET to a DOs disc in drive B:) are now given. It is assumed that the following DOS PLUS commands have been given to prepare the destination disc for the copy.

MD B:\LC
MD B:\LC\H
MD B:\DOCNS

<u>Copy all files from DOS Root Directory</u>

A>MOVE &.DOSPLUS.* -NET B:\ -DOS

<u>Copy all files from DOS Subdirectory \DOCNS</u>

A>MOVE &.DOSPLUS.\DOCNS.* -NET B:\DOCNS -DOS

<u>Copy all files from DOS Subdirectory \LC</u>

A>MOVE &.DOSPLUS.\LC.* -NET B:\LC -DOS

<u>Copy all files with type LIB from DOS Subdirectory \H</u>

A>MOVE &.DOSPLUS.\LC.\H.LIB.* -NET B:\LC\H -DOS

NOTE - when copying from ANFS the source directory must be referred to the URD using the & character because the Currently Selected Directory on ANFS cannot be restored after the MOVE operation.

Copying files between MOS and MOS

In addition to copying files between DOS and MOS, files can also be copied directly from MOS to MOS using either two different filing systems or the same filing system as source and destination. When copying files between MOS filing systems MOVE will only copy files from the specified directory and will not search any filetype subdirectories unless the /C option is supplied. The default operation provides the same type of copy as performed by the ADFS and DFS *COPY command or MOS *MOVE command and is therefore suitable for copying MOS files (ie BBC BASIC programs or VIEW text files). It is also possible to copy an MOS directory that contains DOS files and DOS filetype subdirectories by specifying the /C option in the source filespec. This type of operation is most suitable for copying DOS files that have been saved to MOS by a previous MOVE command.

As an example the DOS files copied to the NET directory DOSPLUS in the previous section could be copied to an ADFS directory $.DOSPLUS on drive 1. The ADFS disc must first be prepared for the copy by issuing the following * commands:

```
*adfs
*dir :1
*cdir dosplus
*cdir dosplus.\lc
*cdir dosplus.\lc.\h
*cdir dosplus.\docns
```

and then giving the following MOVE commands:

```
A>MOVE &.DOSPLUS.* -NET /C :1.$.DOSPLUS -ADFS
A>MOVE &.DOSPLUS.\DOCNS.* -NET /C :1.$.DOSPLUS.\DOCNS -ADFS
A>MOVE &.DOSPLUS.\LC.* -NET /C :1.$.DOSPLUS.\LC -ADFS
A>MOVE &.DOSPLUS.\LC.\H.LIB.* -NET /C :1.$.DOSPLUS.\LC.\H -ADFS
```

If user ERIC on the NET also has directories $.ERIC.BASIC and $.ERIC.VIEW which contain BBC BASIC files and VIEW text files respectively then these can be copied to the same ADFS disc (assuming directories :1.$.BASIC and :1.$.VIEW are present) using these two MOVE commands.

```
A>MOVE &.BASIC.* -NET :1.$.BASIC -ADFS
A>MOVE &.VIEW.* -NET :1.$.VIEW -ADFS
```

Wild Card File Specifications

In the examples given so far a group of files has been specified by the * wild card character which matches all the files in the source directory for both DOS and MOS files. It is also possible to specify that only certain files that match a given pattern of characters will be copied using combinations of the wildcard characters *, ? and #. The wildcard characters have different meanings dependent on the source filing system as given by the following table.

Filing System	Wildcard	No. Characters Matched
DOS	*	0-8
DOS	?	0-1
MOS	*	0-10
MOS	#	1

There are certain differences between the use of wildcards for DOs and MOS filing systems which should be noted. To illustrate this consider the following DOs files that have been copied to a MOS filing system.

A.BAT AA.BAT AB.BAT ABC.BAT TEXT1B.DOC TEXT2B.DOC TESTB.DOC

These files will be saved under MOS with the following names:

BAT.A BAT.AA BAT.AB BAT.ABC DOC.TEXT1B DOC.TEXT2B DOC.TESTB

The following table shows the files matched by the given DOS and MOS wildcard specs:

Filing System	Wildcard	Files Matched
DOS	*.*	All 7 DOS files
DOS	A*.BAT	A.BAT AA.BAT AB.BAT ABC.BAT
DOS	A?.BAT	A.BAT AA.BAT AB.BAT
DOS	TE?TB.DOC	TEXT1B.DOC TEXT2B.DOC TESTB.DOC
MOS	*	all 7 MOS files
MOS	BAT.A*	BAT.A BAT.AA BAT.AB BAT.ABC
MOS	BAT.A#	BAT.AA BAT.AB
MOS	DOC.TE#T#B	DOC.TEXT1B DOC.TEXT2B
MOS	BAT.*A	BAT.A BAT.AA
MOS	DOC.T*B	DOC.TEXT1B DOC.TEXT2B DOC.TESTB
MOS	DOC.T*X*B	DOC.TEXT1B DOC.TEXT2B

From the table it can be seen that ? and # behave differently because ? will match 0 or 1 characters but # will match 1 character only. The * character functions identically for DOS and MOS if it is used as the last character of the filename or filetype. When using MOS wildcard specs the * character does not have to be the last character of the filename or filetype hence it is possible to use patterns such as *A (all files ending in A) and T*B (all files starting with T and ending with B). These patterns are illegal under DOS and will give an error message if used. Care must therefore be taken when copying back from MOS to DOS to ensure that the wildcard spec will copy all the required files. If in doubt about whether all the required files will be copied it is best to use a pattern which will copy more files than required (ie A* not A#) and then delete any files not required.

NOTE - Wildcard filespecs can be used only on MOS filing systems that support the OSGBPB function with A = 8 (Read names from current directory). This includes DFS, ADFS, NFS and ANFS but NOT CFS or RFS. For CFS and RFS filespecs must specify a single file only (ie contain no wildcard characters).

Filename Character Translation

When copying from DOS to MOS the destination filename is formed from the filename and filetype of the source DOS file. In a few special cases this can produce a destination filename which contains invalid characters and therefore cannot be saved to the MOS filing system. To overcome this problem the invalid characters are translated to valid MOS characters to allow the file to be saved. When the file is subsequently copied back from MOS to DOS the invalid characters are translated back again to restore the file's original name. The following table shows the character translation used.

DOS character	MOS character
#	?
$	£
&	{
-	+
@	=

The corresponding MOS characters have been chosen to ensure that no conflict will occur with other non alphanumeric DOS characters that do not require translation (see below).

When copying MOS files from MOS to DOS (ie VIEW text file) the following additional non alphanumeric characters are invalid under DOS and are translated to the underline character _.

```
! \ { } [ ] ; / < >
```

The following non alphanumeric characters are valid under both MOS and DOS.

```
% ' ( ) _ ~
```