

# *Xcopy*

---

## **User's Guide**

---

**Sequential Software, Inc.**

**Release 1.4**

**April, 1996**

© Copyright Sequential Software, Inc. 1987-1997. All rights reserved.  
10 Wilsey Square, Suite 9, Ridgewood, New Jersey 07450 USA  
Phone (201) 652-3300. Fax (201) 652-5596.  
Revision 6 9/97.

# Contents

<b>Introduction</b> .....	<b>5</b>
<b>Installation</b> .....	<b>7</b>
Installing from Tape (3420, 3480/3490, etc.) .....	7
Installing from Diskette or Emailed Package .....	7
To run comparisons before replacing copyfile .....	7
Permanent Placement of XCOPY .....	7
<b>XCOPY Command Syntax</b> .....	<b>9</b>
Options (defaults are underlined): .....	9
Fileid's in the Command Line .....	10
Time and Space Considerations .....	10
Reserved Minidisks .....	11
Moving Minidisks .....	11
<b>Examples</b> .....	<b>13</b>
<b>Options</b> .....	<b>16</b>
<b>Incompatibility Chart</b> .....	<b>31</b>
<b>Messages and Return Codes</b> .....	<b>33</b>
<b>Appendix A: Nucleus Extension Considerations</b> .....	<b>39</b>
<b>Appendix B: Files loaded from the XCOPY installation tape</b> .....	<b>40</b>
Tape File 1: The essential, executable modules .....	40
Tape File 2: The help files. ....	41
Tape File 3: Sample XCOPY applications. ....	41
<b>BLOCKS</b> .....	<b>42</b>
MEASURING FRAGMENTATION .....	43
FILE SYSTEM CONSISTENCY CHECKING .....	45
<b>BLOCKS Messages and Return Codes</b> .....	<b>47</b>
<b>CMPR</b> .....	<b>51</b>
Messages: .....	51
<b>FCOMPARE</b> .....	<b>52</b>
Messages: .....	52
<b>MACCOMP</b> .....	<b>53</b>
Messages: .....	53
<b>RDBLK</b> .....	<b>54</b>
Messages: .....	54
<b>XCTIME</b> .....	<b>55</b>
<b>XCBENCH</b> .....	<b>55</b>
<b>XCCOUNT</b> .....	<b>56</b>
<b>XCF</b> .....	<b>57</b>
Options .....	57
Usage Notes .....	58
Messages .....	58
<b>ZZAP</b> .....	<b>60</b>
<b>Index</b> .....	<b>61</b>



---

# Introduction

XCOPY and the other modules on the installation tape are program products of Sequential Software, Inc. If any questions arise concerning their installation or use, please call us at (201) 652-3300. Our address is

Sequential Software, Inc.  
10 Wilsey Square, Suite 9  
Ridgewood, NJ 07450

XCOPY is a compatible replacement for the COPYFILE command under VM/CMS. It supports all options of COPYFILE at high speed and offers many additional features as well.

## **XCOPY Release 1.4 Notes:**

1. Improved elapsed time performance with nonsynchronous DASD, i.e., RAMAC, 9345's, or anything ESCON-attached. Though we still don't expect performance always to be 100 percent of what's seen with the "good old" synchronous DASD, it should be close. In particular, the FORMAT function again overlaps copying and formatting when output is to a new, nonsynchronous minidisk.
2. Better compatibility with nonsynchronous DASD. In other words, no special zaps are necessary, whatever kinds of disks are used.
3. Better compatibility with the minidisk cache, we think. The overall benefit is rather difficult to determine. Certainly, smaller copies that happen to read the same data several times will be faster.
4. Support for the year 2000 in the XCOPY options that have a date parameter -- AFTER, SINCE, BEFORE, and CDATE. If a 2-digit year is specified, 00-49 is interpreted as 2000-2049. 50-99 is interpreted as 1950-1999. A 4-digit year can be specified. 'AFTER 10/19/03' should have the same effect as 'AFTER 10/19/2003'.
5. A Filelist-like command called XCL which should allow better performance for group copies when the files are selected by means of the full-screen list. Although scores or hundreds of files may be selected to be copied at once through Filelist, XCOPY would view the task as many single file copies. XCL communicates with XCOPY to make it aware at the beginning of all the files to be copied. This often allows XCOPY to use its fast group copy function, which gains considerable efficiency by batching the files together in a certain way.

## **XCOPY Release 1.3 Notes:**

1. Provides the fastest means available to move data stored in CMS files. The high speed is available for all options and for variable length records.
2. Runs in full 31-bit exploitation mode under VM/ESA.
3. Combines disk formatting, checking, and copying into a single, fast operation when moving minidisks. Copies filemode 0 files without requiring write access to the input disk.
4. Allows REXX exits for data modification and file selection and assembler exits for data modification.
5. Performs the same pattern matching as LISTFILE with "\*" and "%" wild characters in the input fileid's. This often obviates the need to first create a CMS EXEC and then repeatedly invoke the COPY command.
6. Selects input files according to several optional criteria: by a date/time range, by whether there is pre-existing output or not, by whether pre-existing output is older or newer than the input, by whether the input is packed or unpacked, and by whether the input is on a R/O extension or not.

7. Optionally erases input files as soon as they are copied. Pre-existing output files can be erased before they are replaced to save disk space.
8. Can copy reserved minidisks.
9. Normally skips packing files that are already packed, or trying to unpack files that are not packed.
10. Detects copying operations which might alter packed files so that they could not be unpacked.
11. Can quickly encrypt files. Encryption following the DES of the National Bureau of Standards is available.
12. Reads filemode 0 files from R/O disks.
13. Previews files to be copied with the NOCOPY option.
14. Capable of scanning for strings anywhere in a record rather than just in the first position. Uppercase and lowercase can be accounted for or disregarded during these scans. Strings can contain embedded blanks or hex data.
15. Writes a CMS EXEC, if desired, containing a list of the output files from a copy operation. This list also can be stacked into the console buffer stack.

---

# Installation

## Installing from Tape (3420, 3480/3490, etc.)

Issue TAPE LOAD to bring down all files from the installation tape. Altogether, they require fewer than 270 1K disk blocks. A full list of the files on the tape with short descriptions is given in Appendix B. Many of the files are demonstration and diagnostic programs. Only the following files are of immediate concern here:

XCMAIN	MODULE	most of the code of the product.
XCMAIN2	MODULE	an extension of XCMAIN used for certain options.
XCOPY	MODULE	a routine to load XCMAIN as a nucleus extension.
XCINST	EXEC	a procedure to initialize a few default settings.
XCTIME	EXEC	two benchmarking aids.
XCBENCH	EXEC	

If it is desired to install help files, issue a second TAPE LOAD after the first one. This will copy over 80 small files from the tape, which require an additional 198 1K disk blocks. A third TAPE LOAD makes available several XCOPY application exits and EXEC's. These are listed in Appendix B.

## Installing from Diskette or Emailed Package

Upload the file "XC14.EXC" to a CMS file, calling it "XC14 EXEC." The file contains only plain ASCII characters. The file upload program should have a simple option to convert to EBCDIC while uploading. Once the file has been loaded, simply run "XC14." This self-contained REXX exec will prompt you for disk mode letters on where to install XCOPY.

Run XCINST EXEC at this point if it is desired to change the upper limit of virtual storage to be used as I/O buffers. This value is set to 256K when the modules come off the installation tape. From 256K to 512K should generally be reasonable. This default storage limit can be altered for particular copies by the VSTORAGE option.

XCINST also can set an I/O limit default in XCF, restrict use of the FM0 option, and change a server userid in XCCOUNT.

## To run comparisons before replacing copyfile

Initially to test XCOPY and to compare it with your current copy utility, such as COPYFILE or FCOPY, the name "XCOPY" can be used to invoke the product. XCTIME EXEC is provided to make performance comparisons with copyfile more convenient. Brief instructions about how to run XCTIME are included at the end of this user's guide. The EXEC invokes COPYFILE MODULE and XCOPY MODULE. The user specifies the particular kind of copy command for which he would like XCTIME to make a performance comparison. There are a great many features and options of copyfile which affect performance, e.g., concatenation, appending, multi-file operation, RECFM, PACK-UNPACK, FROM-FOR, etc. XCBENCH EXEC is provided to run a variety of tests which exercise several of these features. It displays performance results as the tests run and produces a report of all results at the end in a file called XCBENCH LISTING A. XCBENCH is described in more detail later in this manual.

## Permanent Placement of XCOPY

The modules listed above should be stored on a disk generally available to CMS users. Any user who enters

## **NUCXLOAD COPYFILE XCOPY (SYSTEM**

has replaced COPYFILE with XCOPY. For convenience, this command might be placed in an EXEC called by each user's PROFILE EXEC when CMS ipl's. This automatically replaces COPYFILE for all users. The XCOPY module itself is very small and should have practically no effect on virtual storage use.

With VM/SP Release 5, the SYSPROF EXEC facility is provided. This EXEC is executed whenever anyone ipl's CMS. The NUCXLOAD of XCOPY could be placed directly in the SYSPROF EXEC, but with ease of system maintenance in mind, it is better to put the command in an EXEC called by SYSPROF. The installation procedure described above is generally the preferable one. More detailed information concerning installation is available in Appendix A.

To temporarily remove XCOPY as a replacement for COPYFILE enter

## **NUCXDROP COPYFILE COPYFILE**

This drops one or two nucleus extensions that are required to front-end COPYFILE. This command is effective only for the CMS machine that issues it.

After installation, enter the COPYFILE command with the TYPE option to copy one small file. If a message displaying the number of records copied appears, XCOPY has successfully relieved the old COPYFILE. If "XCOPY116I XCOPY not licensed for this CPU" appears, please call Sequential Software. The installation tape provided should have been customized for your machine.

# XCOPY Command Syntax

Use the XCOPY command to efficiently copy, concatenate, append, pack, encrypt, or reformat CMS files. XCOPY is a compatible replacement for COPYFILE and would usually be invoked with that name. All features of COPYFILE are supported at high speed, and many new features are available as well.

The format of the command:

```
XCOPY fileid1 [fileid2 ...] [fileido] [(options ...)]
```

## Options (defaults are underlined):

Type <u>NOType</u>	<u>NEWDate</u> OLDDate	<u>NEWFile</u> REPlace NOCopy	<u>PRompt</u> NOPRompt
FRom recno FRLabel xxxx FRString xxx	FOR numrec TOLabel xxxx TOString xxx TO recno	SKIPRep REPOnly OLDER NEWER	SPecs <u>NOSpecs</u> SAMEDate
CASE { <u>Upper</u>   Mixed   Lower   Ignore }		Fill {c   hh   <u>40</u> }	
OVLY APPend	RECfm {F   V} LRecl nnn	TRUnc <u>NOTRunc</u>	%x
Single	UPcase LOWcase	TRAns EBcdic	SPARSE NOSPARSE
PAck UNPack PPack	<u>PACKWng</u> NOPACKWn	PACKEd UNPACKEd	ERASE PREERase
VStorage {nn   nK}	SON <u>SOFF</u>	<u>NOUPDirt</u> UPDirt	BYpass
AFTer mm/dd/yy [hh:mm:ss] SINCE mm/dd/yy [hh:mm:ss] BEFore mm/dd/yy [hh:mm:ss] CDate mm/dd/yy		ENCrypt xxxx DECrypt xxxx DESEnc xxxxxxxx DESDec xxxxxxxx	NOEXT REORG
EXEC EXECName name	STack LIFO <u>FIFO</u>	FSTEXit filename RECEXit filename RECLRecl nnn	NOSTR STRinit
FORMAT cuu	BLKsize {nn   nK}	FM0	CHEck

---

## Fileid's in the Command Line

The file identifiers on the command line give the full name of a CMS file — filename, filetype, filemode. The last fileid on the right specifies the output file. Only one fileid may be given, in which case it names both the input and output. A valid filemode number should be given or left blank. If a number is specified in an input fileid, it restricts the input file selection. If the output mode number is not given, it is taken from the first input file, or if replacing, from the replaced file.

Wild characters may be used in the input fileid's to perform pattern matching in the same way as the LISTFILE command. "\*" means to match with any string of any size; "%" means to match with any one character. Equal signs (=) can appear in all but the first fileid. An equal sign is evaluated by replacing to the extent possible with the name from the corresponding position in the first fileid. If wild characters appear in the first fileid, ='s are replaced by names from the first file found that matches the pattern.

The fileid's are processed from the command line in two different orders depending on whether there is the possibility of multiple output or not. The possibility of multiple output occurs if and only if the output specification has an "=" in a corresponding position where the first fileid has a wild character. Examples are given in the next section. A single output file can be forced despite how the fileids are entered by means of the SINGLE option. If there is multiple output, wild characters can appear only in the first input fileid. Each output file consists of a concatenation of the set of input files found by processing the input fileids of the command line from left to right. There is one left to right pass and one output file every time a file matches whatever pattern is specified in the first fileid. There are so to speak many horizontal passes within one vertical pass over the first fileid.

If there is single output, wild characters can appear in any of the input fileids. The one output file consists of a concatenation of the files matching whatever pattern is specified in the first input fileid, followed by all files matching the second fileid, and so on for all of the input fileids. There are so to speak several vertical passes, one per input fileid of the command line, within one horizontal pass left to right.

The record format and record length of a new output file is automatically determined from the input files concatenated into it unless the RECFM or LRECL options are used. The automatic determination is as follows. If any input files contain variable length records, the output format is variable. The output lrecl is the greatest of any of the input files. If fixed length records are being written, they are padded when necessary with the FILL character to the output lrecl. When overlaying or appending to an already existing output file, the characteristics of that file are not changed. If such a file is in fixed length format, the input records are padded or truncated if necessary to that length.

## Time and Space Considerations

It is a familiar caveat that many factors affect performance. Just a few of the factors when copying CMS files are input and output disk block sizes, disk device types, hardware configuration, record format, file sizes, the degree of fragmentation of data and free space, the number of files processed, use of concatenation or appending, the amount of available main memory, presence of common options (e.g. FROM-FOR, PACK-UNPACK, REPLACE), and of course, the particular software used. XCOPY attempts to adapt automatically to different circumstances to provide the best performance possible. The following case is a good particular example. When the input and output disks have the same block size, there are many relatively small files, there is no concatenation and no appending, and there are no record-oriented options, then XCOPY uses a mode of operation we will call "fast group copying."

Very fragmented files can be much slower to copy than fairly "smooth" files. XCOPY automatically optimizes to mitigate this effect. The fast group copy mode is also relatively immune to fragmentation. To take advantage of this, one might want to specify a group copy even though only one file would be selected to copy. The BLOCKS command provided with XCOPY can measure the degree of fragmentation. Typical occasions for severe fragmentation are log files which are slowly appended to over a long time and non-filemode 6 files which suffer many random updates by record number. A fast group copy in general will not substantially reduce fragmentation in the output. If defragmentation happens to be the reason for copying, use the REORG option for group copies.

It retards fast group copying enough to insure that all files are "straightened out." All single file copies and group copies with record-oriented options always defragment the output.

Although XCOPY includes special processing to try to provide the best performance possible with the Shared File System introduced in VM/SP6, SFS operations always impose a great penalty in comparison with minidisk operations. The penalty is not so obvious of course if the utility under scrutiny was already slow on minidisks.

If serious quantities of data, say many megabytes, are to be moved, it is probably worthwhile to consider briefly whether XCOPY will have plenty of memory available for I/O buffers. If the default buffer size chosen at installation time is relatively small, or one is uncertain about what that value might be, specify the option VStorage 512K. This instructs XCOPY to attempt to obtain 512K for I/O buffers. Another 108K is required for work space. If this much memory is not available, XCOPY reduces the buffer space sufficiently so it can run. There is little practical benefit to specifying VStorage greater than 512K. Although a 1-megabyte virtual machine would usually be quite large enough to allow XCOPY to stretch out, memory can disappear quickly. It is allocated and fragmented in many ways, e.g., for accessed disk directories and nucleus extensions (such as XCOPY). Moving whole minidisks is the kind of job which often goes better when one insures there is adequate memory available. It may involve many cylinders of data and hundreds or thousands of files. One thousand files on the output disk require over 64K of new directory space. A very large new disk directory could easily spill into prime user storage by the end of the copy operation. This could make it impossible to execute some programs in the user area. XCOPY takes measures to avoid this problem during common group copies. If the problem seems to occur anyway, one should try releasing the output disk at the end of the copy, and reaccessing it if necessary to reallocate the directory in memory.

## Reserved Minidisks

Reserved minidisks can be moved if a group copy is specified, i.e., a command in the form

```
XCOPY * * B = = C.
```

The RESERVE command has filled these disks with one, large file. No free space remains even to support the normal double directory scheme of the CMS file system. For this reason, normal file copying procedures fail with the "disk full" error when trying to move the reserved disk file. In its group copying mode, XCOPY recognizes reserved disks and is able to copy them by means of some special handling. A single file copy still results in the disk-full error.

## Moving Minidisks

Beyond its file and record oriented features, XCOPY moves whole minidisks significantly faster than any other utility. It is usually at least four times faster than DDR and does not have the restrictions concerning device types and relative sizes of the input and output disks. This common, important function deserves some particular comments. Certain options of XCOPY are often associated with moving minidisks and will be discussed in that context below. They are FM0, FORMAT, OLDDATE, VSTORAGE, and CHECK. The typical form of the XCOPY command when it is used to move minidisks:

```
XCOPY * * B = = C ( FM0 FORMAT outcuu OLDDATE VS 512K
```

The VSTORAGE option is less critical than the others, but when moving minidisks, it is probably worthwhile to insure plenty of I/O buffer space. There are very likely many buffers of data to be shoveled. It is also more likely that I/O overlap is possible, i.e., simultaneous reading and writing. XCOPY will automatically attempt I/O overlap when appropriate. Since the buffer space is then divided into reading and writing halves, it is best to provide a little more space to begin with. Occasionally, it might be preferable to retard XCOPY a little for the sake of overall

system response time. This can be done by restricting its buffer space, say, to 256K or less.

FM0 insures that all files, i.e. including filemode 0, will be copied, even when the input disk is R/O. This option has no effect if the input disk is accessed R/W.

FORMAT would not be necessary if the output disk were already formatted, but this is generally not the case when moving minidisks. XCOPY will combine the formatting and copying work so that there is only one pass over the output disk. This is considerably faster than doing a separate, preliminary format followed by the copy. This would be true even if the XCF command distributed with XCOPY were used to perform the formatting. The output disk for XCOPY is specified actually by the outcuu parameter to the FORMAT option rather than the output filemode. At the end of the copy, the output disk will be accessed with the specified output filemode. The usual two questions asked by the CMS FORMAT command ("are you sure" and about the disk label) will appear. When moving many disks, one would usually want to arrange somehow to stack the two replies in an exec. As a default, the output disk block size is made the same as that of the input disk. It can be set explicitly through XCOPY's BLKSIZE option, but there is a considerable performance advantage when the input and output block sizes are the same.

OLDDDATE is specified because the default for XCOPY is to give the current date and time to new output files, which is rarely what would be desired when moving minidisks.

CHECK is only recommended if a file system consistency problem is suspected. It integrates a consistency check with the copying operation. It incurs little overhead, but the general likelihood of a file system problem does not warrant making CHECK a standard option when moving minidisks. XCOPY often moves groups of files at once. This technique can propagate some file system errors from input to output disk which would be corrected by slower, file-by-file copies. Such errors would not be exacerbated, only carried over. The CHECK option allows the significant errors to be corrected with only a small effect on performance. The description of the CHECK option gives more details about these kinds of errors. Some errors of course cannot be corrected by any copying procedure. With CHECK, messages are displayed to inform of any detected errors. Irretrievably broken files are skipped and at least omitted from the output disk. The CHECK option should only be used when copying whole minidisks. The BLOCKS command distributed with XCOPY can also perform an independent file system consistency check on a disk, or a particular file. It inspects for a few more kinds of error conditions which are less critical.

The preceding comments apply to moving minidisks, as opposed to Shared File System directories. FM0 does not apply to SFS input, nor does FORMAT to SFS output. XCOPY probably moves SFS files as fast or faster than any other utility, but SFS operations are much slower than minidisk I/O. This may not be apparent if one only deals with a handful of data blocks. It should also be borne in mind that most of the I/O work occurs in the SFS server rather than in the machine where the initiating command was entered. Of course with a small perverse effort, minidisk I/O can be brought to a crawl as well. The limit to what is possible though is much lower for SFS directories.

---

## Examples

XCOPY MY FILE A YET ANOTHER A	Makes a second copy of MY FILE under the name YET-ANOTHER.
XCOPY FLUFF STUFF A == (PACK XCOPY FLUFF STUFF A (PACK	These two are equivalent, the FLUFF STUFF file is packed unless it is in packed format already.
XCOPY MY * A VM = T	Copies all of the MY files from the A-disk to the T-disk giving them all the new filename of VM.
XCOPY * * A == T	Copies all files from disk A to disk T.
XCOPY MY * A MY KLUDGE T	Concatenates all of the MY files into one.
XCOPY MY * A = KLUDGE T	Also concatenates all MY files into MY KLUDGE T.
XCOPY * *ED B (UNPACK	Unpacks all files on the B-disk which have a filetype ending with ED.
XCOPY * %%% A = DONE B	All files on the A-disk with a 3-character filetype are copied to the B-disk and given the filetype DONE.
XCOPY ABC* * T = = D	All files on the T-disk which have a filename beginning with ABC are copied to the D-disk.
XCOPY %%BIG* *STUFF* A OLD = B	Copies files with BIG in positions 3-5 of the filename and with STUFF anywhere in the filetype to disk B always with the output filename OLD.
XCOPY AA B%%* * 123=45= CC F	The input files are selected from all accessed disks, have the filename AA, and filetypes of at least 3 characters beginning with a B. All input is concatenated into one output file, 123AA45A CC F.
XCOPY * XEDIT S TF T T = XX T	Copies each XEDIT file to an XX file appending TF T T to the end of each one.
XCOPY * XEDIT S * EXEC A BIG LUMP T	Concatenates all XEDIT S files followed by all EXEC A files into one file.
XCOPY FLUG * * = STUFF =	Can invite a problem by specifying "=" for the output mode when the input mode is variable. If FLUG FILE A existed, it would be copied to FLUG STUFF A. This output file would then also qualify as input to be copied to FLUG STUFF A, which already exists. Either avoid specifying input and output filemodes in this manner, or use the SKIPREP option.
XCOPY ABC TEXT * ALL TEXT A (SKIPREP	Collects all ABC TEXT files on all accessed disks into one file. If ALL TEXT A already exists, no copy occurs.
XCOPY ABC* TEXT * = = A (SKIPREP	The first ABC* TEXT files found in the disk search order are brought to the A-disk. This is a multiple output case because

there is an \* in the input filename and an = in the output filename. Note that when there is multiple output and SKIPREP, the meaning of \* for the input mode is to copy matching files only from the first disk in the search order rather than from all disks.

XCOPY TERA DATA A MORE DATA B (SPECS LRECL 60 NOPR /10A/ 1 1-25 4 71-80 46

Reformats records according to the second line, which XCOPY reads. The resulting output records are in the following format.

position:	contents:
1-3	the constant '10A'
4-28	input rcd positions 1-25
29-45	blanks, the default FILL
46-55	input rcd postions 71-80
56-60	blanks, the FILL character

XCOPY \* ASSEMBLE M FUNCTION DESCS P (FRS FUNCTION: FOR 10 T APP

Selects 10 records from each ASSEMBLE file on the M-disk beginning in each file at the first record containing the string 'FUNCTION:'. Each group of 10 records is appended to the end of FUNCTION DESCS P if it already exists. The name of each input file displays at the terminal as the copy from it completes.

XCOPY MY EXEC A NEW EXEC A (RECFM V TRUNC

Converts the input to variable length records without trailing blanks.

XCOPY \* REVIEW A (ENC 9CrI

Packs and encrypts all REVIEW files using the key '9CrI'.

XCOPY \* REVIEW A (DEC 9CrI

Decrypts and unpacks all REVIEW files. *PLEASE NOTE* that forgetting the key is quite similar to erasing the files.

XCOPY \* \* F == B (AFTER 00:00

All files on the F-disk dated after the beginning of the current day are copied to the B-disk.

XCOPY \* \* F == B (AFTER 6/1/87 ERASE

All files on the F-disk dated after the beginning of 6/1/87 are moved to the B-disk. They are erased from the F-disk.

XCOPY \* %%%\* F == B (AFTER 12/15/86 17:30:00 BEFORE 1/16/87 FSTEX MYEX

Selects input files according to the following.

- on F-disk
- filetype of at least 3 characters
- dated within the specified range
- accepted by REXX EXEC — MYEX XCOPY

Selected fileids are passed last to the FST exit, which may delete, accept, or insert any fileid.

XCOPY FLORID PROSE A MY MURK U (RECEX CONVERTR

Passes each record to the REXX EXEC — CONVERTR XCOPY. This exit can accept, modify, delete, or insert before each record passed to it.

XCOPY \* COBOL C == B (UNPACKED CDATE 5/22/87 PACK

All unpacked COBOL files on the C-disk which have the date given are packed to the B-disk.

XCOPY \* \* A == T (AFT 6/8/87 EXEC

All files dated from the specified time to the present are copied

XCOPY ** C == B (AFT 6/20 FORMAT 2B1	from the A to the T-disk. A list of the files is written to CMS EXEC A.
XCOPY TST* * A == T (UNPACK FORMAT 197 BLK 2K	Minidisk 2B1 is first formatted and accessed as the B-disk. All files dated June 20 or later in the current year are then copied from the C to the B-disk. The block size of the output disk will be the same as the C-disk.
XCOPY ECLECTIC FILE N == M (PREERASE	The 197 disk is formatted at 2K and accessed as T. The selected input files are unpacked to the T-disk.
XCOPY ALL NOTEBOOK A TEMP FILE A (SON FRS 'Go Blue' TOS =====	If the file already exists on the M-disk, it is erased before the copy.
XCOPY MY TXTLIB A TEXT RCDS R (SON FRS X'41A0303058F0' SOFF TOS ""	The lines from the first one containing 'Go Blue' to just before the next one containing '=====' are copied. SON causes the single quotes not to be interpreted literally as part of the search string. The quotes are needed to make the blank between 'Go' and 'Blue' part of the string.
XCOPY ** Q == V (NEWER OLDD REPO T NOCOPY	The from-string is given in hex. SON causes X'41A0...' not to be taken as literal EBCDIC data. SOFF turns off this quoted string mode so that the to-string can be literally four single quotes.
	Names of files on the Q-disk which also exist on the V-disk and have a more recent date on the Q-disk are typed at the terminal. No copying actually occurs because of NOCOPY. OLDD has no effect until NOCOPY is dropped and the copy is actually performed. The V-disk might be R/O.

---

# Options

## Type

displays at the terminal after each input file is copied, a message in the following format:

```
nnnn records copied from fileidi to fileido (new file).  
                                or      (replace).  
                                (append).  
                                (overlay).
```

If no indication of the status of the output file appears in parentheses at the end of the message, then the status is unchanged since the last indication. Message XCOPY714I is also displayed indicating the XCOPY release number.

## NOType

suppresses the display of the TYPE option of the names of files being copied. The verifying redisplay of an encryption or decryption key is also suppressed.

## NEWDate

uses the current date as the creation date of new files.

## OLDDate

uses the date of the input file as the creation date of the output file. If a set of input files is concatenated to one output file, the date is taken from the first input file copied.

## NEWFile

prevents overwriting of any already existing files. Before any data is copied, a search is made for an output file with the same name as what is specified to XCOPY for output. If one is found, the copy terminates with an error message. This is the default except when the output file is the same as the input.

## REPlace

allows already existing files to be overwritten. This is the default when only one fileid is given or the output specification is “= =,” i.e., the output file is the same as the input. The process of replacement is to first write the output under the temporary name COPYFILE CMSUT1, erase the file to be replaced, and then rename the temporary file. Note that if the copy output and the file to be replaced will not both fit on the output disk, this option will not help.

## PRompt

displays messages requesting that additional information be entered beyond what was given on the command line. SPECS and TRANS are options which require such additional information.

## NOPRompt

inhibits prompting messages from the SPECS and TRANS options; a read is still performed for the necessary information. This option is useful when XCOPY is invoked from an EXEC which stacks the information. The EXEC hardly needs the prompt, which would only become distracting clutter on the screen. If both the SPECS and TRANS options are present, the first prompt will be for the SPECS information.

## SPecs

reformats records according to a list that one is prompted to enter at the console. The prompt is

*XCOPY601R Enter specification list:*

This message is suppressed by the NOPROMPT option. The expected response is one or more pairs of operands in the following format. The first operand indicates in one of three ways the data to appear in the output record. The second operand gives the byte position, or column number, where the data should be in the output record.

```
bbb-eee  
/string/   col  
hxx...
```

bbb-eee are the beginning and ending byte positions of a section of data in the input record. If eee is greater than the input record length, the end of the record is used as the ending position. The first position is numbered 1.

string is any string of uppercase or lowercase characters or numbers delimited by any non-alphameric character (such as "/").

hxx... is an even number of hexadecimal digits preceded by the letter "h".

col is the number of the byte position in the output record where the data indicated by the first operand is to appear. The first position is numbered 1.

To enter more than one line of specification pairs, add two plus signs (++) at the end of a line. This signals XCOPY to do another console read. Basically, there are two kinds of reformatting provided by SPECS — (1) selection and repositioning of parts of the input record, and (2) overlaying with constant data at specified positions. If two specification pairs overlap in the output record, i.e, there are two determinations of one output position, the latter specification takes precedence. Output positions which are not determined at all are filled with the FILL character, which defaults to a blank. The FILL option is available to change this character. When the OVLY option is in effect along with SPECS, any output columns not determined by the SPECS list are filled not by the FILL character, but by data from the pre-existing output record in these columns. For example if the following two lines are entered,

```
xcopy * assemble a (specs ovly noprompt  
/1.0/ 69
```

the effect is to place the string '1.0' in columns 69-71 of each record of each file on the A-disk with filetype ASSEMBLE. All other positions in these records are unchanged.

Fixed-length output records are either padded with the FILL character or truncated to that fixed length. SPECS fields beyond this length are ignored. If an output record length different from the input length is desired, include the LRECL option. Variable-length output records always contain all the SPECS fields and nothing more beyond the last field.

## **NOSPecs**

indicates no record reformatting according to a specification list.

## **OVly**

overlays an existing output file record by record with data from the input file or files. The output file must already exist or else XCOPY will stop with an error message. To the extent possible, data from the pre-existing output record is used in place of the FILL character. This may be useful when the input and output files are the same and when reformatting records with the SPECS option since with OVLY, only the changing record positions must be specified. The entire output record need not be specified in the SPECS list since the unchanging parts of the record will be filled automatically. The output file will contain the same number of records after the OVLY operation as before, regardless of the amount of input. Without SPECS, the input records flop one by one over the output records.

## **APpend**

appends input data to the end of the output file. If the output file does not already exist, APPEND has no effect on the copy. Records are truncated or padded with the FILL character and the format is changed between fixed or variable if necessary to agree with the output file. If the output is variable length records, the input records are not truncated or padded; the maximum lrecl of the output file may increase.

## **RECFm { F | V }**

converts the input to fixed or variable length record format. The default is to write the output in the same format as the input. If concatenated input files have mixed formats, the default without this option is to produce variable length output. RECFM is incompatible with APPEND. When converting from fixed to variable, trailing blanks or the FILL character are only removed when the TRUNC option is specified as well. If a file is simply converted from fixed to variable, it may occupy additional data blocks because in the variable format an extra two bytes per record are embedded in the data blocks to store the record length.

## **LRecl nnn**

sets the output record size to nnn. Fixed length records are truncated at the end or padded with the FILL character if necessary. This option has no effect for variable length output records.

## **TRUnc**

removes trailing blanks, or whatever the current FILL character is, from variable length output records. This option has no effect for fixed length output.

## **NOTRunc**

indicates that trailing blanks, or whatever the current FILL character is, are not to be removed from variable length output records.

## **FILL { c | hh }**

specifies a character used with several options. The default is a blank, X'40'. The FILL character is specified either directly by a single character following the FILL option, or by two hexadecimal digits. It sets the padding character used when it is necessary to extend fixed length records. It is the trailing character removed by TRUNC. It fills the gaps between SPECS fields. It is what is assumed for best compression during the PACK operation to be the most frequently occurring byte value.

## **EBcdic**

translates Binary Coded Decimal (BCD) characters to Extended Binary Coded Decimal Interchange Code. The following conversions occur: < to ), & to +, % to (, # to =, @ to ', and ' to :

## **UPcase**

converts all lowercase letters to uppercase.

## **LOWcase**

converts all uppercase letters to lowercase.

## **TRAns**

translates characters according to a list of pairs of characters. Unless NOPROMPT is in effect, one receives the following prompt for the list.

*XCOPY602R Enter translation list:*

A character is entered in the list either by directly typing one character or by means of a two-byte entry which is interpreted as the hexadecimal representation of one character. A double plus sign (++) at the end of a line of character pairs signals XCOPY to issue a console read for another line. When the first character of a pair is encountered during copying, the second character of the pair is substituted for it. For example, when the following lines are entered,

```
xcopy my file a test out t (trans noprompt  
* - A f0 00 ff
```

all occurrences in MY FILE of \*, the character A, and X'00' appear in TEST OUT as respectively -, X'F0', and X'FF'. If any of the options EBCDIC, UPCASE, or LOWCASE are used with TRANS, and they present a conflict about how a certain character should be translated, the TRANS specification takes precedence.

## **Single**

forces all input into one output file even if the specification of the fileid's on the command line would indicate multiple output files. If there would have been multiple output files, the single file has the name of what would have been the first output file.

## **ERASE**

erases input files after they are copied. The copy operation might be called more precisely a "move" with this option. In the event of a disk full condition, an HX entered, or an abnormal termination, those files which have been copied successfully will be erased from the input disk.

## **ENCrypt key**

encrypts output based on a key entered on the command line following this option. The key may be up to 4 bytes long. Output will appear to be random data. Please note that it will indeed be worthless data if the key is forgotten. Input data is automatically packed before it is enciphered with the key. This not only saves space, but also makes the encryption more secure. This encryption technique is not intended to foil the most determined codebreakers, although it probably would if applied successively two or more times with different keys. Encryption according to the DES is available with the DESENC option described below. ENCRYPT is much quicker than DESENC. Note that the original unencrypted data blocks still reside on disk, but are marked as free and are not associated with a file. Again, encrypting two or more times in succession can be more secure in this respect as well. To be sure that encryption is performed with the intended key, the key is redisplayed unless the NOTYPE option is used.

## **DECRypt key**

operates much like the ENCRYPT option. It reverses the effect of ENCRYPT when run with the same key. Please note that DECRYPT will effectively encrypt the data again if it is run with a different key.

## **DESEnc key**

encrypts following the Data Encryption Standard published by the National Bureau of Standards. The encryption key is entered on the command line after this option. The key can be up to 7 bytes long. For a given amount of data, this technique requires much more processing time than the ENCRYPT option. It should be mentioned that the National Bureau of Standards does not recognize a software implementation of the DES algorithm.

## **DESDec key**

reverses the encryption of the DESENC option when provided with the same key.

## **EXEC**

writes a list of the names of the output files from a copy operation in a file called CMS EXEC. The format of the file is the same as seen when LISTFILE is run with the EXEC option.

## **EXECName name**

has the same effect as the EXEC option except the file written is “name EXEC” rather than “CMS EXEC.”

## **STack**

pushes into the CMS console buffer stack the fileids of the output files. By default, the order in which the filenames are stacked is FIFO, first-in first-out.

## **FIFo**

indicates the order in which fileids are to be stacked by the STACK option is first-in first-out. This option implies the STACK option so that STACK might be omitted if FIFO is specified.

## **Lifo**

indicates the order in which fileids are to be stacked by the STACK option is last-in first-out. This option implies the STACK option so that STACK might be omitted if LIFO is specified.

## **NOUPDirt**

postpones updating of the output disk directory until all files have been copied to the disk. This is a considerable performance advantage when many smallish files are copied. This is the default except in certain cases in which the directory is automatically updated to conserve disk space. These cases involve multiple output files and either the REPLACE option, or the ERASE option when the input disk is the same as the output. In these situations, erasing from a disk is interleaved with writing of new files. Specifying the NOUPDIRT option suppresses the multiple directory updates in these cases as well. The degree of risk in specifying this option is usually small but must be borne in mind — the chance that some hardware failure, lightning, etc. would prevent the final update of the disk directory. Program checks and hx are not a problem; they still permit the final update. NOUPDirt might be specified when the user wants the performance improvement and knows that there is not any risk, that the disk is temporary scratch space or easily recoverable.

## **UPDirt**

updates the CMS disk directory after closing each output file. It is the automatic default only for a couple of particular cases described above under NOUPDIRT. There normally should be no need to specify this option. XCOPY usually performs one directory update per output disk, and this should be sufficient. When copying many files, continual directory updates can cause considerable performance degradation. There can easily be more work copying the directory than copying data files.

## **AFTer mm/dd/year hh:mm:ss**

selects input files with a date and time after what is specified by this option. There is some flexibility in the format of the date and time. The basic form is shown above. Either the date or the time can be given by itself. The default for the time is the beginning of the day, 00:00:00. The default date is the current day. When both date and time are specified, the date should come first. If the first 2 digits of the date are greater than 12, the format is automatically assumed to be YEAR/MM/DD. If only MM/DD is given, YEAR defaults to the current year. If a 2-digit year is specified, 00-49 is interpreted as 2000-2049. 50-99 is interpreted as 1950-1999. A 4-digit year can be specified. 'AFTER 10/19/03' should have the same effect as 'AFTER 10/19/2003'. If only HH:MM appears, SS is taken to be 00. When just HH is specified, MM:SS is assumed to be 00:00. AFTER can be used with the BEFORE option to select files from a limited range in the past.

## **BEFore mm/dd/year hh:mm:ss**

selects input files with a date and time before what is specified by this option. The different forms in which the date and time limit can be given are described above under the AFTER option. BEFORE can be used with the AFTER option to select from a limited time range.

## **SINCE**

is the same as the AFTER option.

## **BYpass**

continues a copy operation after a read error. This helps in salvaging unaffected files from a disk with some bad "spots," either magnetic or in the file system.

## **CDate mm/dd/year**

input files are selected which have dates the same as what is specified by this option. The format of the date can be varied somewhat as described under the AFTER option.

## **FRom recno**

begins copying in each input file at the specified record number. The default is to begin at the first record, number 1.

## **FOR numrec**

halts copying in each input file after processing the specified number of records. The default is to copy until the end of each file.

## **FRLabel string**

postpones the commencement of copying for each input file until the string is found at the beginning of an input record. This record is the first one copied. The string may be up to 32 characters long. It is translated to uppercase before any comparisons unless different processing is requested through the CASE option. If SON appears in the command line before this option without an intervening SOFF, the string must be enclosed by either single quotes or X'...'. Single quotes

permit embedded blanks in the string. X'...' allows hex data; only the hex digits 0-9 and A-F should be used, two per byte.

## **FRString string**

functions exactly as FRLABEL except that the entire record length is scanned for the string rather than just the first position. Embedded blanks or hex data are allowed only in quoted string mode, which is turned on by the SON option.

## **TOLabel string**

stops copying for each input file if and when the character string is found at the beginning of an input record. The record containing the string is not copied. The string may be up to 32 bytes long. It is as a default translated to uppercase before any comparisons. This can be altered with the CASE option. If the TOLABEL string were an initial substring of a FRLABEL string specified in the same copy, then this would prevent any output, and therefore this situation is indicated to be an error by message XCOPY172E. The string is specified as described under FRLABEL.

## **TOSTring string**

functions in exactly the same way as TOLABEL except that a scan for the string is made through the whole input record rather than in just the first position. If the TOSTRING string is contained anywhere in a FRSTRING or FRLABEL string specified in the same copy, then this would prevent any output, and therefore this situation is indicated to be an error by message XCOPY172E.

## **TO recno**

stops copying each input file at the specified record number. The record with this number is the last included in the copy.

## **CASE { Upper | Mixed | Lower | Ignore }**

indicates how uppercase and lowercase translations are handled with respect to string searching required by FRLABEL/FRSTRING and TOLABEL/TOSTRING. UPPER, the default, converts the string being scanned to uppercase before the search. This assumes the string will appear in uppercase in the file. MIXED or LOWER leave the string unchanged before the search. IGNORE allows the string to match with disregard for uppercase or lowercase. The processing for IGNORE is as if both the string and the records to be scanned were converted to uppercase before any compares.

## **PAck**

compresses a file to save disk space. The compression technique is essentially the following. Two or more consecutive occurrences of the FILL character are replaced by one byte containing a few bit flags and a duplication count. The default FILL character is a blank. This can be altered by the FILL option. Four or more consecutive occurrences of any non-FILL character are condensed to three bytes. Packed files should not be modified because otherwise it would be unlikely they could be unpacked. XCOPY will not modify packed files in a way which could preclude their unpacking unless the NOPACKWN option is selected. If an input file is already packed, XCOPY ignores the PACK option and copies the file unchanged. Use the PPACK option to force a second packing. Packed files are always stored as fixed-length records with Irec1 1024 on EDF disks and Irec1 800 on the older CDF disks.

## **PPack**

packs the input data whether it is already packed or not. The default with the more usual PACK option is to copy without repacking a file that has already been packed.

**UNPack**

restores a file compressed by PACK to its original, expanded form. If a file had been packed twice, it would have to be unpacked twice as well to uncompress it. If this option is used in a multi-file copy, and an input file is not packed, XCOPY copies the file as it is and proceeds with the next file.

**PACKWng**

causes a copy to end with an error if a packed input file is encountered, and the current options specified could modify the file so it could not be unpacked. This option is normally in effect as the default.

**NOPACKWn**

disables the check for modifications to the data format of a packed input file which could make it impossible to unpack the data.

**PACKEd**

selects only packed input files for copying.

**UNPACKEd**

selects only unpacked input files for copying.

**SKIPRep**

skips copies which would replace already existing output files. The effect might often be viewed as copying all files which have not already been copied. This option is similar to NEWFILE. The difference is that the current input is just skipped instead of ending with an error when an output file would be replaced. An important feature of this option is that it can change how specification of "\*" for an input filemode is interpreted. The usual meaning for XCOPY is to look for input files on any and all disks. With SKIPREP, the effect of an "\*" filemode is to process only the first occurrence of the input file or files in the disk search order.

**REPOnly**

performs only copy operations which replace already existing output files. This is the inverse of SKIPREP. In essence, the output files select the input files.

**NEWER**

replaces an already existing output file on the condition that the input file has a more recent date and time. An older input file is skipped. If there is no output to replace, the input is always copied.

**OLDER**

functions like NEWER except that the date comparison goes the other way. A replace occurs only when the input file is older than the output file. A newer input file is skipped.

**SAMEDate**

selects input files much like OLDER and NEWER except the date and time of the input file must match that of the output file exactly for the output to be replaced.

## VStorage { n | nK }

adjusts the upper limit for the amount of main memory allocated for I/O buffers. This overrides a default upper limit provided at installation time. XCOPY tries to allocate as much as possible within the limit unless the memory requirement for a simple copy is known exactly beforehand and is small.

## %x

sets “x” instead of “%” as the place-holding wild character which may appear in the input fileid’s of the XCOPY command. This option is used in the same way as in the LISTFILE command.

## NOEXT

prevents normal searching of read/only extension disks to find input files. The normal procedure when there are no wild characters in the input fileid, and the file is not found on the specified disk, is to look on any and all read/only extensions of the originally specified disk. The first occurrence of the file on any of these other disks is used. When there are wild characters in either the filename or the filetype, but not the filemode, all matching files on read/only extensions of the specified disk are copied as well as matching files on that disk. This is as if an “\*” had been given for the filemode, but only the subset of all accessed disks represented by the read/only extensions to the specified disk are searched.

## FSTEXit filename

The FSTEXIT option is provided to allow user-defined criteria to select which input files to use during a copy. This facility helps keep the XCOPY command line simple, while permitting flexibility for inserting files not generated by the command line, and for deleting (excluding) files that are generated by the command line. Exits may be written in either the REXX or EXEC2 languages, and follow a format similar to the REEXIT facility. The FSTEXIT is loaded by XCOPY from the first occurrence of “filename XCOPY” on an accessed disk. It remains resident and active during the entire copy operation. EXEC variables created by the exit are available on subsequent calls to it.

Each time the exit is called by XCOPY, a 64-byte FST will be available to it under the name “FST” (or “&FST” for EXEC2). An exception is the final call to the exit for each output file, when variable “FST” is set to zero length. The format of an FST is provided in several IBM manuals, such as “VM/SP CMS Command and Macro Reference”, SC19-6209 and “VM/SP Data Areas and Control Block Logic”, LY20-0891.

The file specified in the FST is that which is to be copied next. The statement “RETURN” for REXX or “&RETURN” for EXEC2 must be used to return control back to XCOPY with one of the following parameters:

“ACCEPT”:	use this file in the copy operation.
“DELETE”:	do not use this file in the copy operation.
“INSERT”:	insert the file defined by variable “FID” which the exit must set. The format of “FID” must be “filename filetype filemode.”
“STOP”:	stop execution of the copy operation.

### Sample FSTEXIT

```
/* **** */
/* this fstexit selects files with an lrecl = 80 */
trace off
lrecl = c2d(substr(FST,33,4))
if lrecl = 80 then RETURN ACCEPT
      else RETURN DELETE
/* **** */
```

### **RECEXit filename**

The RECEXIT option is provided to allow user-defined criteria to select and alter records in a file during a copy. XCOPY provides many options for changing the records in a file, and the RECEXIT extends this by allowing a user to code a specific routine to modify a file with an assembler language routine or a REXX or EXEC2 language exec. The search order for a RECEXIT is:

- 1- if *filename* XCOPY exists on any accessed disk then that file is loaded as an exec exit, either interpretable or compiled.
- 2- if *filename* MODULE exists on any accessed disk then that file is LOADMODed and assembly language conventions apply.
- 3- an OS SVC 8 is issued to load *filename* as either a TEXT file or from a GLOBAL TXTLIB search chain. If this fails, then a system abend results; otherwise, assembly language conventions apply.

### **Interpreted Exec Exit Conventions**

The exec is loaded and remains resident during the entire copy operation but does not always remain active. The exit does remain active for all the records in a file and any files specified for concatenation, i.e., for each output file. This means that EXEC variables created by the exit are available on subsequent calls to it for that set of files. However, after each output file is written, the exit will be stopped and all variables defined by the exit will be deallocated. The GLOBALV command can be used by the exit to pass information from one invocation to the next.

There are two variables available to the RECEXIT when it is called by XCOPY: "XRECORD" and "FST". The first is set to the record currently being processed by XCOPY. The exit may pass it back as is, modify it, delete it, or insert a record before it. XRECORD is set to null length after the last input record has been given to the exit for each set of concatenated input. "FST" is set to the FST of the file from which this record came. Note that this variable is similar to, but independent of the same variable name that is used in the FSTEXIT facility.

The record passed to the exit will have been decrypted or unpacked depending upon whether those options were specified on the command line. Other data modification options (SPECS, RECFM, LRECL, TRANS/UPCASE/LOWCASE/EBCDIC, PACK, ENCRYPT, DESENC) will not yet have changed the record; such transformations are performed on the records that the RECEXIT returns to XCOPY. Control is given back to XCOPY by using "RETURN" for REXX or "&RETURN" for EXEC2 with one of the following parameters:

<b>“ACCEPT”:</b>	use XRECORD as passed back from the exit.
<b>“DELETE”:</b>	delete the current record.
<b>“INSERT”:</b>	insert XRECORD as created by the exit. The next call to the exit will have the same record passed to it as on this call.
<b>“STOP”:</b>	stop execution of the copy.

### Sample Exec RECEXIT

```

/*****
/* copy only those records with "1.0" in columns 69-71*/
trace off
if XRECORD = " then RETURN ACCEPT /* end of file */
txt1 = substr(XRECORD,69,3)
if txt1 = "1.0" then RETURN ACCEPT
                else RETURN DELETE
/*****

```

### **Compiled Exec Exit Conventions**

Compiled execs work in much the same manner as interpreted execs. Additional statements in the exit are required to retrieve and give back records to XCOPY. This method of writing an exec is compatible when running interpreted. The exec exit is called just once for each set of input files being written to a single output file, with the logic for retrieving records from XCOPY imbedded within the exec. The following table describes the statements needed to process records in a compiled Rexx record exit.

<b>“Address XCEXITR GET”:</b>	Variable XRECORD will be set to the current record. Variable FST is set to the to the FST of the file from which this record came.
<b>“Address XCEXITR GIVE”:</b>	The exit must set a special variable called “XRC” to either ACCEPT, DELETE, INSERT, or STOP, as described in the box above. This indicates the action to be taken on the current value of XRECORD.

The following example shows the logic required by a compiled Rexx record exit.

### Sample REEXIT for Compiled Execs

```
/* **** */
/* copy only those records with "1.0" in columns 69-71 */
Address XCEXITR GET /* Set XRECORD to first record */
Do While XRECORD /= "
txt1 = substr(XRECORD,69,3)
if txt1 = "1.0" then XRC = "ACCEPT" else XRC = "DELETE"
Address XCEXITR GIVE; Address XCEXITR GET
End
/* **** */
```

### **Assembler Exit Conventions**

The assembler exit is located and loaded as described previously and remains resident during the entire copy operation. On each call to the exit registers are set as follows:

GPR 0: points to an extended parameter list with the current record pointed to by the argument list (the second and third words of the EPLIST).

```
DC A(0)
BEGARG DC A(record)
ENDARG DC A(record end + 1)
```

GPR 1: points to a 2 word parameter list:

```
DC A(current record) "the current record address"
DC A(current FST)
```

GPR 12: has the entry point address of the exit.

GPR 13: has the address of an 18 word save area.

GPR 14: contains the return address in XCOPY.

GPR 15: has the entry point address of the exit.

For variable length records ENDARG-BEGARG yields the length of the current record. On the last call to the exit for each set of concatenated input files, the current record address will be zero. On return to XCOPY, GPR 15 should be set with one of the following codes with the corresponding meaning:

- 0: use whatever the current record address points to.
- 4: delete the record just passed to the exit.
- 8: no more records are to be copied to the current output file.
- 12: insert the record pointed to by the current record address.
- 16: halt the entire copy operation with a non-zero return code.

For variable length records, GPR 0 must have the length of the record being passed back to XCOPY, for action codes 0 and 12. In general, the record may be modified in place provided that its length does not increase for variable length records. The exit must preserve GPR 13 and return control to the address found in GPR 14.

## **RECLRecl nnn**

allows the length of records returned from an exit to be different than the input length. For fixed length records, nnn is the exact number of bytes in the records; for variable length, nnn defines the maximum size record that can be created by the REEXIT. Use this option if an exit changes the length of fixed length records or increases the maximum size of variable length records.

## **REORG**

insures during fast group copies that all fragmented files are “straightened out.” In a fragmented file, the order of the data blocks on disk varies considerably from their logical order in the file. Fragmentation can be, but normally is not a significant problem under CMS because of how file system I/O is performed and because of the usual segregation of data on separate minidisks. Normally, XCOPY reads data blocks in logical order and writes them in available free space. This “defragments” the data. During a group copy of many files between disks with the same block size, XCOPY can provide a large performance advantage by using a mode of operation in which the files are not necessarily defragmented. This mode of fast group copying is used automatically as the default when possible. The REORG option is available to alter this default for group copies in which one would like to insure that all files are defragmented. It generally sacrifices some of the speed advantage.

## **NOSTR**

prevents the STRINIT function during XCOPY initialization. STRINIT resets core allocation pointers for user storage. XCOPY performs this function when it is invoked in the typical way from a command line. Depending on how well the previous command cleaned up, the STRINIT may be necessary for XCOPY to allocate enough memory. When XCOPY is invoked from another program, say via SVC 202, rather than from a command line, it skips the STRINIT in order to preserve allocations by the invoking program. The NOSTR option causes this storage initialization to be skipped as well when a copy is initiated by a command line. This may be useful on rare occasions to save data in user storage during an XCOPY execution.

## **STRinit**

forces the STRINIT function when XCOPY would otherwise have skipped it as described above.

## **BLKsize {nn | nK}**

specifies in conjunction with the FORMAT option the disk block size at which the output disk should be formatted. If FORMAT appears without BLKSIZE, the output block size is the same as that of the input disk. Valid block sizes are 512, 1K, 2K, and 4K.

## **FORMAT cuu**

combines copying and minidisk formatting into a single, faster operation. In typical cases, only one pass over the output disk is necessary to accomplish both formatting and copying. BLKSIZE is an option associated with FORMAT which allows specification of the output block size. The default without it is to use the same block size as that of the input disk. The output disk is specified by the CUU following the FORMAT option, rather than by the filemode of the output file. The output disk will be accessed as the mode given in the XCOPY output fileid. The XCF command must be available when the FORMAT option is used. XCF is described later in this manual.

In general, there are few incompatibilities between FORMAT and the other

features of XCOPY. The input and output fileid's can be specified in the usual variety of ways. Practically all of the usual options are available, such as AFTER-BEFORE, FSTEX, PACK, RECFM, etc. XCOPY will internally perform 2 separate passes over the output disk for formatting and copying when the block size is changed or a record-oriented transformation (such as PACK or RECFM) is specified. These cases will be slower than those requiring only one pass, but they are still significantly faster than executing the CMS FORMAT command followed by a copy. XCOPY internally invokes the CMS FORMAT command to initialize the file system, but performs most of the I/O itself. The familiar questions ("are you sure?" and about the disk label) are still displayed and must be answered.

## **PREErse**

operates much like REPLACE except that the space occupied by the file being replaced is freed before the copy rather than afterwards. This may allow some copies to complete which would have failed because of insufficient disk space. It must be borne in mind that if the copy does fail for any reason after initial syntax checking, the pre-existing output file has already been erased.

## **SON SOFF**

allow embedded blanks or hex data in the search strings specified for the options FRLABEL, TOLABEL, FRSTRING, and TOSTRING. As the command line is scanned from left to right, SON and SOFF turn a "quoted string mode" on and off. They affect how the command line is parsed. They can appear an indefinite number of times in the command line. In quoted string mode, the search strings must be enclosed by single quotes, or by X' ... '. In the latter case, the data is specified in hex — 0-9 and A-F, 2 hex digits per byte. The default is SOFF.

## **NOCopy**

suppresses any output in order to allow a preview of the files which will be copied. Input file selection and output name formation still occur. The files which would be copied can be displayed with the EXEC, EXECNAME, STACK, or TYPE options. NOCOPY affords a kind of special directory scanning utility when used with the file selection capabilities of XCOPY, such as fileid pattern matching and the options REONLY, SKIPREP, NEWER, OLDER, FM0, NOEXT, AFTER, BEFORE, CDATE, SAMEDATE, PACKED, and UNPACKED. For instance, SKIPREP can display the fileids which two disks do -not- have in common; REONLY can display the ones they share. REONLY and NEWER would display among the shared files ones which are more recent on the first disk, etc.

## **SPARse**

prevents blocks of all binary zeroes from being written to disk. Only zero index pointers to them are created instead. This can save disk space. When reading, the file system automatically provides a data block of binary zeroes when it encounters a zero pointer. In its fast group copy mode, XCOPY never expands nor condenses blocks of all zeros. Prior to VM/SP6, COPYFILE would only write a sparse file when the input was sparse and the lrecl equaled the block size. In SP6, the basic file system was changed to automatically condense out zero blocks. This applies to recfm V files as well as recfm F. Prior to SP6, sparse recfm V files were not valid. The SPARSE option essentially is the default for XCOPY under SP6 to be compatible with this change in the file system. The data block count for a sparse recfm F file reflects the 'saved' blocks, the count for a sparse recfm V file does not.

## NOSPArse

causes all output zero disk blocks to be allocated and written, which insures that the output file can be processed by any VM release. The CMS file system in VM/SP6 was changed to automatically condense out zero disk blocks and create sparse files whenever possible. Some of these SP6 sparse files are invalid and cannot be read in prior VM releases. If they can be read, the time spent in I/O may increase.

## FM0

permits filemode 0 files to be copied from a R/O disk. This option is primarily useful when moving all files from one disk to another. It allows an entire disk to be moved without requiring write access to it, which otherwise would be necessary to read any filemode 0 files on it. If the input disk is accessed R/W, or does not contain any filemode 0 files, this option has no effect. There is occasionally also a modest performance benefit from FM0. When the allocation bit map of the input disk is available, XCOPY will inspect it to help optimize performance during certain minidisk moves. These moves are the ones in which the output disk is either empty or is being formatted during the copy by use of the FORMAT option. XCOPY does not obtain the bit map for a R/O disk unless FM0 is specified. In general, when migrating minidisks, the typical basic options which should be specified would be FORMAT, FM0, and OLDDATE.

FM0 is not restricted to just moving minidisks. It can be used any time one would like to include filemode 0 files on a R/O disk in the input file selection. 'XCOPY \*\* B0 == A (FM0 OLDD' would in fact retrieve only the filemode 0 files on the B-disk. 'XCOPY \*\* B0 (FM0 TYPE NOCO' would just list these files. The primary purpose of filemode 0 files is to prevent file sharing problems when users each have the same name for their own peculiar files, e.g., PROFILE XEDIT, ALL NOTEBOOK, etc. Any protection filemode 0 provided has been mostly an illusion, since it was always fairly easily circumvented. This option can be restricted to Class C and E users by customizing XCOPY with the XCINST exec.

## CHECK

performs file system consistency checking during group copies with very little extra overhead. For most of the common kinds of group copies, XCOPY attempts by one technique or another to move several files at-a-time. This can provide performance improvements of several hundred percent. Some kinds of file system errors are propagated from input to output disk by these techniques. The errors are not exacerbated, but they are carried over. With only a small compromise in performance for a fast group copy, the CHECK option detects and, whenever possible, corrects such errors during the copy. For example, files which share blocks on the input disk will not on output. Unreadable files with, say, bad block numbers will be skipped, much as with the BYPASS option. Problems in the allocation bit map are resolved with the CHECK option. These are either allocated blocks which are not used by any file, or more critically, unallocated blocks which are used by a file. These errors are generally not common and do not warrant constant checking. If one has the least suspicion though of a file system integrity problem on a disk to be copied, CHECK can be recommended. Messages are displayed to inform of whatever problems are found.

The BLOCKS command distributed with XCOPY independently performs similar file system checking. It checks some additional, but less critical aspects of the file system.

# Incompatibility Chart

<u>Option</u>	<u>Incompatible Options</u>
AFTER	CDATE
APPEND	CHECK, DECRYPT, DESENC, DESDEC, ENCRYPT, LRECL, NEWDATE, NEWFILE, NEWER, OLDER, OLDDATE, OVLY, PACK, PPACK, RECFM, REPLACE, REONLY, SKIPREP, UNPACK
BEFORE	CDATE
BYPASS	NOBYPASS
CDATE	AFTER, BEFORE
CHECK	APPEND, DECRYPT, DESDEC, DESENC, EBCDIC, ENCRYPT, FOR, FRLABEL, FROM, FRSTRING, LOWCASE, LRECL, NOCOPY, NOSPARSE, OVLY, PACK, PACKED, PPACK, RESEXIT, RECFM, REORG, SPECS, TO, TOLABEL, TOSTRING, TRANS, TRUNC, UNPACK, UNPACKED, UPGASE
DECRYPT	APPEND, CHECK, DESDEC, DESENC, ENCRYPT, EBCDIC, FROM, FRLABEL, FOR, LOWCASE, LRECL, OVLY, PACK, PPACK, RECFM, SPECS, TOLABEL, TO, TRUNC, TRANS, UPGASE
DESDEC	APPEND, CHECK, DECRYPT, DESENC, ENCRYPT, EBCDIC, FROM, FRLABEL, FOR, LOWCASE, LRECL, OVLY, PACK, PPACK, RECFM, SPECS, TOLABEL, TO, TRUNC, TRANS, UPGASE
DESENC	APPEND, CHECK, DECRYPT, DESDEC, EBCDIC, ENCRYPT, FROM, FRLABEL, FOR, LOWCASE, LRECL, OVLY, RECFM, SPARSE, SPECS, TOLABEL, TO, TRUNC, UNPACK, TRANS, UPGASE
EBCDIC	CHECK, DECRYPT, DESENC, DESDEC, ENCRYPT, PACK, PPACK, UNPACK
ENCRYPT	APPEND, CHECK, DECRYPT, DESENC, DESDEC, EBCDIC, FROM, FRLABEL, FOR, LOWCASE, LRECL, OVLY, RECFM, SPARSE, SPECS, TOLABEL, TO, TRUNC, UNPACK, TRANS, UPGASE
ERASE	FM0, NOCOPY
FM0	ERASE
FOR	CHECK, DECRYPT, ENCRYPT, PACK, PPACK, TOLABEL, TOSTRING, TO, UNPACK
FORMAT	NEWER, NOCOPY, OLDER, REPLACE, REONLY, SKIPREP, SAMEDATE
FROM	CHECK, DECRYPT, DESENC, DESDEC, FRLABEL, PACK, UNPACK, PPACK, FRSTRING, ENCRYPT
FRLABEL	CHECK, DECRYPT, DESENC, DESDEC, ENCRYPT, FROM, FRSTRING, PACK, PPACK, UNPACK
FRSTRING	CHECK, DECRYPT, DESENC, DESDEC, ENCRYPT, FROM, FRLABEL, PACK, UNPACK, PPACK
LOWCASE	CHECK, DECRYPT, DESENC, DESDEC, ENCRYPT, PACK, PPACK, UNPACK
LRECL	APPEND, CHECK, DECRYPT, DESENC, DESDEC, ENCRYPT, PACK, PPACK, UNPACK
NEWDATE	APPEND, OLDDATE
NEWER	APPEND, FORMAT, NEWFILE, OLDER, SAMEDATE, SKIPREP
NEWFILE	APPEND, NEWER, OLDER, OVLY, REPLACE, REONLY, SKIPREP
NOCOPY	CHECK, ERASE, FORMAT, NOPACKWN, PACKWNG, PREERASE
NOPROMPT	PROMPT
NOSPARSE	CHECK, SPARSE
NOSPECS	SPECS
NOSTRINIT	STRINIT

NOTRUNC	TRUNC
NOTYPE	TYPE
NOUPDIRT	UPDIRT
OLDER	APPEND, FORMAT, NEWER, NEWFILE, SAMEDATE, SKIPREP
OVLY	APPEND, CHECK, DECRYPT, DESENC, DESDEC, ENCRYPT, NEWFILE, PACK, PPACK, REPLACE, REONLY, SKIPREP, UNPACK
PACK	APPEND, CHECK, DECRYPT, DESDEC, EBCDIC, FOR, FRLABEL, FROM, LOWCASE, LRECL, OVLY, PPACK, RECFM, SPARSE, SPECS, TO, TOLABEL, TRUNC, UNPACK, TRANS, UPCASE
PACKED	CHECK, PPACK, UNPACKED
PACKWNG	NOPACKWN
PPACK	APPEND, CHECK, DECRYPT, DESDEC, EBCDIC, FOR, FRLABEL, FROM, LOWCASE, LRECL, OVLY, RECFM, SPARSE, SPECS, TO, TOLABEL, TRUNC, UNPACK, TRANS, UPCASE
PREERASE	APPEND, NOCOPY, OVLY, SKIPREP
PROMPT	NOPROMPT
RECEXIT	CHECK
RECFM	APPEND, CHECK, DECRYPT, DESDEC, DESENC, ENCRYPT, PACK, PPACK, UNPACK
REORG	CHECK
REPLACE	APPEND, FORMAT, NEWFILE, OVLY, SKIPREP
REONLY	APPEND, FORMAT, NEWFILE, OVLY, SKIPREP
SAMEDATE	APPEND, FORMAT, NEWER, NEWFILE, OLDER, SKIPREP
SKIPREP	APPEND, FORMAT, NEWER, OLDER, OVLY, REPLACE, REONLY
SPARSE	DESENC, ENCRYPT, PACK, PPACK
SPECS	CHECK, DECRYPT, DESENC, DESDEC, ENCRYPT, NOSPECS, PACK, PPACK, UNPACK
STRINIT	NOSTRINIT
TO	CHECK
TOLABEL	CHECK, DECRYPT, DESENC, DESDEC, ENCRYPT, FOR, PACK, PPACK, TOSTRING, TO, UNPACK
TOSTRING	CHECK, DECRYPT, DESENC, DESDEC, ENCRYPT, FOR, PACK, PPACK, TO, TOLABEL, UNPACK
TRANS	CHECK, DECRYPT, DESENC, DESDEC, ENCRYPT, PACK, PPACK, UNPACK
TRUNC	CHECK, DECRYPT, DESENC, DESDEC, ENCRYPT, NOTRUNC, PACK, PPACK, UNPACK
TYPE	NOTYPE
UNPACK	APPEND, CHECK, DESENC, EBCDIC, ENCRYPT, FOR, FROM, FRLABEL, LOWCASE, LRECL, OVLY, PACK, PPACK, RECFM, SPECS, TO, TOLABEL, TRANS, TRUNC, UPCASE
UNPACKED	CHECK, PACKED
UPCASE	CHECK, DECRYPT, DESENC, DESDEC, ENCRYPT, PACK, PPACK, UNPACK
UPDIRT	NOUPDIRT

---

## Messages and Return Codes

**XCOPY002E {input | overlay} file “fn ft fm” not found. RC=28**

*The specified input or overlay file does not exist.*

**XCOPY003E Invalid option “option”. RC=24**

*The displayed option is either misspelled, not an abbreviation, or is not a valid option.*

**XCOPY024E File “fn ft fm” already exists — specify “REPLACE”. RC=28**

**XCOPY025E Options would change format of packed file “fn ft fm”. RC=25**

*The input file is in packed format and a selected option would probably make it un packable. To override this check use the NOPACKWN option.*

**XCOPY029E Invalid parameter “parm” in the option “option” field. RC=24**

**XCOPY030E File “fn ft fm” already active. RC=28**

*An input file is currently open for writing.*

**XCOPY037E Disk “mode” is read/only. RC=36**

*The output disk is not writeable.*

**XCOPY042E No fileids specified. RC=24**

**XCOPY048E Invalid mode “mode”. RC=24**

*The displayed filemode is not correct. It must be a valid CMS filemode.*

**XCOPY054E Incomplete fileid specified. RC=24**

*FILENAME, FILETYPE, and FILEMODE must all be specified for the input and/or output fileid.*

**XCOPY055E More than 8 characters specified in a fileid: “string”. RC=24**

*CMS allows only 8 characters for a FILENAME or FILETYPE.*

**XCOPY062E Invalid character “x” in fileid “fn ft fm”. RC=20**

*“x” is not an allowed character in a CMS fileid.*

**XCOPY063E No {translation | specification} list entered. RC=40**

*When TRANS or SPECS is selected, XCOPY reads more information from the console.*

**XCOPY064E Invalid “string1” at or near “string2”. RC=24**

*An incorrect translation or specification value has been entered. Please see the description of either the TRANS or SPECS option in this manual.*

**XCOPY065E “option” option specified more than once. RC=24**

**XCOPY066E “option” and “option” are conflicting options. RC=24**

*Please find a list of incompatible options elsewhere in this User's Guide.*

**XCOPY067E Concatenated input files are illegal with the “option” option. RC=24**

*Options ENCR, DECR, DESE, DESD PACK, UNPACK may not be used with the concatenation feature.*

**XCOPY068E File fileid is not packed. RC=32**

**XCOPY069E Disk “mode” not accessed. RC=36**

*Use the CMS ACCESS command to activate the selected disk, or use a different filemode.*

**XCOPY101S “Specs” temp string storage exhausted. RC=88**

*A large number of SPECS entries have been given and an internal work area has filled up. Call Sequential for help.*

**XCOPY114E command error “nn”. RC=nn**

*(The front-end cannot nuxload XCMAIN.)”command” may be ‘nuext’, ‘fsread’, or ‘dmsfree’. An fsread error might be a result of releasing the disk containing XCMAIN MODULE. A dmsfree error would generally indicate that free storage is in severe short supply.*

**XCOPY116I XCOPY not licensed for this CPU.**

**XCOPY117I XCOPY license expires in less than 2 weeks.**

**XCOPY118E XCOPY license expired. RC=1**

*If any of the previous three messages appear, please call Sequential to obtain a functional version of the product.*

**XCOPY156E FROM nnn not found — file “fn ft fm” has only “mmm” records.**

**RC=32**

*The FROM value specified exceeds the total number of records in the input file.*

**XCOPY157E Label “label” not found in file “fn ft fm”. RC=32**

*The FRLABEL label specified is not found in the input file.*

**XCOPY172E FRLABEL/FRSTRING and TOLABEL/TOSTRING data preclude any output. RC=24**

*The TO string is found in the FROM string.*

**XCOPY173E No records were copied to output file “fn ft fm”. RC=40**

*Another message should accompany this explaining why no records were written. This message only appears when the TYPE option is in effect.*

**XCOPY601R Enter specification list:**

*This is the prompt when SPECS is requested. Please respond by entering the character the record reformatting information described in this manual under the SPECS option. The NOPROMPT option suppresses this message.*

**XCOPY602R Enter translation list:**

*This is the prompt when TRANS is requested. Please respond by entering the character translation information described in this manual under the TRANS option. The NOPROMPT option suppresses this message.*

**XCOPY700E XCOPY has run out of main memory. Increase virtual storage. RC=700**

*XCOPY needs more contiguous storage to run in. Releasing disks, nuxdrop-ing some extensions, and typing HX may free up sufficient storage for XCOPY. Alternatively, DEFINE more storage and re-IPL.*

*Certain options and/or record formats can require more main memory.*

**XCOPY701I File “fn ft fm” is not packed but will be copied as is.**

*UNPACK has been selected but the input file is not in packed format. This message appears only when copying many files with a wildcard specification.*

**XCOPY702I File “fn ft fm” is already packed but will be copied as is.**

*PACK has been selected and the input file is already in packed format. This prevents the double packing of files which does not reduce the file size. Specify PPACK to pack a file more than once.*

**XCOPY703I {encrypt | decrypt} password being used is “word”.**

*The password being used for encryption or decryption is displayed to confirm what was specified. If the password is lost, then the file cannot be decrypted. This message might be valuable if one enters inadvertently one password, but is thinking of a different one. Use the NOTYPE option to suppress this message.*

**XCOPY704E Error “nn” during reading of exit “fn ft”. RC=nn**

*An error was encountered reading the exit from disk into memory. The FSREAD macro lists the possible return codes.*

**XCOPY705E Exit file not found - “fn ft”. RC=28**

*The FSTEXIT or RECEXIT specified is not on a currently accessed disk.*

**XCOPY706E Not enough virtual storage to load exit — “fn ft”. RC=706**

*More free storage can be obtained by releasing disks, nucxdrop-ing extensions, and HX-ing, or DEFINE more storage.*

**XCOPY707E Error “nn” trying to establish SUBCOM environment. RC=nn**

*Contact Sequential Software if this message appears. An error has occurred with either an FSTEXIT or RECEXIT.*

**XCOPY708E Record length=“len” from recexit, but must be “len”. Record #: “nnn”. RC=708**

*The length of a record returned by the record exit is not correct. If attempting to redefine the record lengths, then use the RECLRECL option.*

**XCOPY709E Call to XCEXIT with invalid parameter. RC=709**

*Contact Sequential Software if this message appears. An error has occurred with either an FSTEXIT or RECEXIT.*

**XCOPY710E Returned value from exit - “value” is not valid. RC=710**

*An exit has passed back an incorrect return code.*

**XCOPY711E Problems with shared variable “FID” whose value is “value”. RC=711**

*Contact Sequential Software if this message appears. An error has occurred with either an FSTEXIT or RECEXIT.*

**XCOPY713E Incorrect record length=“len” from RECEXIT. Record #:“nnn”.**

*The length of a variable length record returned by a record exit is either negative or too large.*

**XCOPY714I “site identification” XCOPY Rel. release number**

*This message identifies the version of XCOPY running and is produced with the TYPE option.*

**XCOPY715E Problems deciphering file “fn ft fm”. Probable cause is an incorrect key. RC=715**

*Either the DECRYPT or DESDEC option has been selected and an error has occurred while decrypting the file. An incorrect key has probably been entered. Re-check the key that was used when encrypting the file and note that the key can contain mixed case letters.*

**XCOPY716I Data block count discrepancy. IB=bbb, IEX=xxx, BM=mmm.**

*A possible file system error has been detected on the input disk or XCOPY has some internal error. The files have probably been copied successfully and the FCOMPARE command should be used to verify this. Save the input disk and call Sequential Software if further assistance is needed.*

**XCOPY717I Data block count too high in file “fileid”. DBC=bbb, RECS=rrr.**

*The specified file has an inconsistency in that the block count (from the FST) is larger than the (item count \* Irecl) / disk-blocksize. The file is copied without the extra blocks and this message is informative in nature.*

**XCOPY718E File “fileid” is not encrypted. RC=718**

*Decryption was specified by the DECRYPT or DESDEC option, but the input file was not encrypted.*

**XCOPY719E “XCMAIN2 MODULE” not on an accessed disk. RC=719**

*XCMAIN2 is automatically loaded as a supplemental nucleus extension when certain options are specified, such as PACK, SPECS, RECFM, TRANS, TRUNC. The module must be available on some accessed disk to use these features.*

**XCOPY720E Module mis-match between XCMAIN & XCMAIN2. Please update XCOPY release. RC=720**

*XCOPY modules may be accessed which are from a different release of the product than the one which was accessed when XCOPY was first executed and loaded. Modules from different releases might have been mixed on the same disk.*

**XCOPY721I nnnn records copied from fileidi to fileido (“output disposition”)**

*This message is produced for each file copied if TYPE is specified. It indicates the number of input records copied and whether the output file is new, replaced, appended, or overlaid. If the output disposition does not appear, it is the same as for the previous file copied.*

**XCOPY722E XCF MODULE is required to FORMAT a disk. RC=722**

*When the FORMAT option of XCOPY is specified, XCF is called; so, XCF MODULE must be available on an accessed disk. XCF is distributed with XCOPY. As an independent command, it operates as a faster replacement for the CMS FORMAT command. It functions somewhat differently when called by XCOPY, but is necessary to complete combined copying and formatting.*

**XCOPY723E Error from DIAG 24 for device “cuu”. RC=723**

*The disk device to be formatted probably is not defined or attached.*

**XCOPY724E Device “cuu” cannot be formatted. RC=724**

*The device is not DASD or it is a 2314 or 3340, which are not supported by the XCOPY FORMAT option.*

**XCOPY725E Filemode “mode” not valid with FORMAT. RC=725**

*If the input filemode is “\*”, the output filemode cannot be “=” when the FORMAT option is specified.*

**XCOPY726E Error “nn” from SUBPOOL “poolname”. RC=nn**

*The displayed error code was returned from the SUBPOOL macro.*

**XCOPY727E CSL routine “DMSxxxx” error “retcode, reason code”. RC=retcode**

*One of the plethora of Callable Service Library routines has given a non-zero return code. If a subsequent message or the explanation of the reason code in the Application Development Reference for CMS does not immediately solve the problem, please contact Sequential Software.*

**XCOPY728I Fast group copy code cannot complete. Code: nn**

*Typical group copies for which the block size does not change and there are no record-oriented options are usually processed by XCOPY in a special mode to improve performance. If an I/O error or file system consistency problem is encountered during such a group copy, XCOPY retries the copy in a slower mode. It is likely though that the slower processing will fail for the same reason. The reason code displayed in the message might be useful to Sequential in determining the cause of the problem. Please call Sequential for assistance.*

**XCOPY729E Parameter missing following “option” option. RC=24.**

*Some options require a parameter to follow immediately after the option keyword. Some examples are FILL, FORMAT, and BLKSIZE.*

**XCOPY730E DMSTRK error, RC = nn.**

*The CMS system routine which allocates and deallocates disk blocks gave the displayed nonzero return code. The most common problem DMSTRK would have in general is insufficient disk space, RC = 8. If XCOPY is operating properly, this should never be seen.*

**XCOPY731E Device type for disk ‘fm’ not determined. RC=8.**

*Although the disk is accessed, a diagnose to determine its device type has failed. This should be a rare circumstance. Has the disk possibly been detached but not released?*

**XCOPY732E Error ‘nn’ from internal routine RDFILE. RC=nn.**

*This is an internal consistency problem which would require a call to Sequential.*

**XCOPY733I File "fileid" has a bad file structure.**

*The file's index blocks are inconsistent or in error. An attempt will be made to copy any other files that match the input specification. The BLOCKS command with the VALIDATE option can help pinpoint the file system error. Please call Sequential for further information.*

**XCOPY734W File "fileid" is sharing block nn with another file.**

*The underlying CMS file system has a consistency error, that could result in serious problems. Two files are sharing the same disk block, nn. XCOPY will attempt to copy both files, correcting the share problem. However, one file probably has incorrect data in it.*

**XCOPY735E Reserved minidisks cannot be resized.**

*An attempt was made to copy a reserved minidisk to an output disk of a different size.*

**XCOPY904T Unexpected UNPACK error at “addr”. RC=256**

*An error has occurred while unpacking a file. The file has either been modified to make it unpackable or a logic error has occurred in XCOPY. If the latter possibility seems to apply, please call Sequential for assistance.*

**XCOPY905E Disk “mode” is full. RC=256**

*The output disk does not have enough available free blocks to complete the copy operation.*

**XCOPY906E Error “nn” processing file “fn ft fm”. RC=nn**

*A read or write error has occurred. The error code is from either a DIAGNOSE 20, A8, or an FSREAD or FSWRITE call to CMS. The file has not been copied. Please call Sequential for assistance.*

**XCOPY907I 13 or 3 - I/O error. 12 - EOF encountered. 25 - insufficient memory.**

*This message may appear after XCOPY906E to document the most common I/O error codes.*

**XCOPY999E XCOPY Internal Abend mmmm(nnn).**

*An internal consistency check has failed. This message is usually followed by an operation exception. Please save the input disk being copied and call Sequential Software.*

**XCOPY1184E File “fileid” not found or not authorized. RC=8**

*An attempt was made to access a file in a Shared File System directory. The file either does not exist, or authorization for it is not available.*

**XCOPY1258E You are not authorized to write file “fileid”. RC=8**

*Shared File System authorization to write the named file is not available.*

**XCOPY1259E Filepool “filepoolid” is full. RC= 40**

*The disk space of the filepool is exhausted.*

**XCOPY1260I File space warning threshold reached or exceeded.**

*Your individual limit of space in the output Shared File System filepool has almost been reached.*

**XCOPY1261E Attempt to exceed maximum 4K filepool blks allowed this user. RC=40**

*The Shared File System space limit for this user is insufficient to complete writing the output file.*

**XCOPY1262E Duplicate object found in filepool catalog. RC=8**

*An attempt was made to write a new SFS file, but it already exists. This error indicates in essence that an earlier directory lookup in XCOPY must have failed; so, a call to Sequential is in order to research the confusion.*

**XCOPY1263E “DMSxxxxx” CSL fast path init failure. RC = nn**

*This probably indicates an invalid CSL parameter list built by XCOPY. Please call Sequential. The XCOPY return code is the same as that from the CSL call.*

---

## Appendix A: Nucleus Extension Considerations

XCOPY MODULE provided on the installation tape is a small program which loads XCMAIN as a nucleus extension and then branches to it. This small program can execute either as a transient or as a nucleus extension itself. The name which it gives to the extension it loads is the name by which it was invoked. For example, entering the command XCOPY results in a nucleus extension called "XCOPY." If XCOPY MODULE is renamed to COPYFILE MODULE (pre-SP5), the nucleus extension it loads will be named "COPYFILE." In this latter case, when the COPYFILE command is entered on subsequent occasions, XCMAIN receives control directly. The nucleus extension containing XCMAIN disappears at any abend or HX. Whenever it is not present, the COPYFILE command will first invoke the small XCOPY, which transparently reloads the nucleus extension before completing the copy function.

Under VM/SP Release 5 in which COPYFILE became part of the CMS nucleus, the small XCOPY MODULE must run as a nucleus extension to replace the original COPYFILE. This is required by the CMS command search order. The following command substitutes XCOPY for the original COPYFILE.

```
NUCXLOAD COPYFILE XCOPY (SYSTEM
```

This command creates a small nucleus extension called COPYFILE. Subsequently, after the first execution of a COPYFILE command, it will be normal to have two nucleus extensions with the name COPYFILE. This technique works well under Release 4 also; so, although other options are available under Release 4, it is recommended that the same technique be used simply for upward compatibility. Prior to Release 5, renaming XCOPY MODULE to COPYFILE MODULE would cause the small XCOPY program to run from the transient area rather than as a nucleus extension. Otherwise, execution is as described above.

---

## Appendix B: Files loaded from the XCOPY installation tape

### Tape File 1: The essential, executable modules.

BLOCKS	MODULE	a command to analyze fragmentation and to perform a file system consistency check.
BLOCKS	DOC	a description of the BLOCKS command.
CALOC	MODULE	diagnostic tool which displays the amount of allocated DMSFREE storage.
CMPR	MODULE	comparator to find differences between two files.
DMSCPY	TEXT	allows the DIRMAINT product to use XCOPY rather than the original COPYFILE when this text is accessed by DIRMAINT before the original text.
FCOMPARE	MODULE	fast comparison program to verify that two files are identical.
FINDIT	MODULE	a maintenance aid which searches for data in memory.
HEXMOD	MODULE	tool to aid in application of maintenance from Sequential Software.
MACCOMP	MODULE	utility to compress MACLIB's quickly.
RDBLK	MODULE	transient which displays FST's and disk blocks.
WRBLK	MODULE	updates disk blocks for debugging and support purposes only. Caution!
XCBENCH	EXEC	runs tests and prepares a report comparing the performance of two or more COPYFILE replacement products. The tests exercise a variety of features.
XCF	MODULE	a faster replacement for the FORMAT command.
XCINST	EXEC	procedure to set default limit for I/O buffer size.
XCMAIN	MODULE	the essential code of the product.
XCMAIN2	MODULE	an extension of XCMAIN.
XCOPY	MODULE	convenient transient to load XCMAIN as a nucleus extension.
XCCOUNT	MODULE	front-ends COPYFILE to measure usage of that command.
XCCREP	EXEC	generates a report on COPYFILE usage from data collected by XCCOUNT.
XCSERV	EXEC	starts up the XCSERV program in a service virtual machine.
XCSERV	MODULE	receives IUCV/MSG information sent to it by XCCOUNT; runs in a service virtual machine.
XCTIME	EXEC	demonstration tool which compares the performance of XCOPY and COPYFILE for user-specified types of copies.
ZZAP	MODULE	another maintenance tool which applies zaps to modules.

## **Tape File 2: The help files.**

XCOPY        HELPMENU    plus its many attendant HELPX COP files.

## **Tape File 3: Sample XCOPY applications.**

ASCIEX	ASSEMBLE	Record exit converts from EBCDIC to ASCII.
EBCDICEX	ASSEMBLE	Record exit converts from ASCII to EBCDIC.
UNCRLF	ASSEMBLE	an exit which removes carriage-return-linefeeds.
CALLCOB	ASSEMBLE	an interface to a COBOL XCOPY exit.
COBSUB	COBOL	a sample COBOL XCOPY exit.
SEARCH	EXEC	an example of using the next two exits to search files for strings.
STRSCAN	MODULE	an exit used by SEARCH for case sensitive character string scanning.
SCANIGN	MODULE	an exit used by SEARCH for case-ignore scanning.
EXPLODE	ASSEMBLE	an exit to expand nested /INCLUDE records.
MRGCHG	MODULE	an exit to apply a change file created by CMPR. This routine in a sense accomplishes multiple FROM-FOR's during a copy and additionally allows insertions at multiple points. Cf. the CMPR command.
UNCAT	DOC	a description of a simple archiving scheme implemented by the next four routines.
CAT	EXEC	calls XCOPY to create or append to an archive.
CATPICK	XEDIT	uses the archive index, XEDIT, and XCOPY to retrieve files.
UNCAT	MODULE	retrieves all files from an archive, or those marked in the index.
NUMINX	MODULE	used by CAT and XCOPY to create the archive index.

The assemble sources and exec's listed above usually contain more detailed comments.

---

# BLOCKS

The BLOCKS command performs the following functions concerned with analyzing and verifying CMS minidisks or files on them.

- Measures the fragmentation of a minidisk or a file.
- Validates the file system with respect to a file, group of files, or the whole minidisk.
- Corrects some file system errors.
- Captures the FST and pointer blocks of a CMS file in a separate file.
- Produces a map of disk block usage.

```
BLOCKS fileidi [ fileido ] [ ( options [ ] ) ]
```

## Options

### DATA

causes data blocks to be read as well as index blocks during fragmentation analysis. If an output fileid is specified to capture the index blocks, these data blocks will also be written to the output file following the index blocks. The DATA option most often would be useful when capturing the disk directory or allocmap in a file. It also must be used with the FRAGMAP option when the allocmap is analyzed to determine the fragmentation of the whole disk. If not all data blocks can fit in the allocated main memory, the remaining blocks are simply ignored.

### NODISP

suppresses the display of fragmentation information at the terminal. It might be useful when it is only desired to capture information in an output file

### INTerval nn

specifies the interval of disk blocks for which data block counts will be accumulated during fragmentation analysis. The default size is 128. This may be either too large or too small to give the best distribution map of block counts. This option allows the interval size to be adjusted.

### FRAGMAP

must be specified to obtain fragmentation analysis of a whole disk rather than one file. The DATA option must also be specified along with FRAGMAP for this kind of analysis, and the input file must be XXX ALLOCMAP fm.

### VALidate

requests file system consistency checking for a disk, a file, or a group of files. To check an entire disk, the input fileid must be '\* \* fm'.

### MAP

generates a simple map of disk block usage in the file BLOCK MAP A. Validate must also be specified.

### FIX

repairs, when possible, file system consistency problems discovered during VALIDATE processing. After validation without FIX, a message will be displayed about whether any discovered errors are fixable or not. These kinds of errors are generally allocated but unused blocks, or used but not allocated blocks. The fixing process involves updating the allocmap in memory. The disk should be reaccessed R/W before running with the FIX option to insure that the allocmap in memory matches the one on disk.

## VStorage {nnK | nnM }

allows the default allocation of main memory by BLOCKS to be altered. The default is about 384K. If this much free storage is not available, specifying a smaller VSTORAGE may allow BLOCKS to run. If the default allocation is not large enough to hold both the disk directory and all of the index blocks of the largest file on the disk, VSTORAGE also allows BLOCKS to allocate more memory.

The various options can be viewed in two groups — those which concern fragmentation analysis and those which concern file system validation. The options of the first group are only effective when the VALIDATE option is *not* specified, and those of the second group apply only when VALIDATE *is* specified. The fragmentation options are described first in what follows.

## MEASURING FRAGMENTATION

The BLOCKS command can measure the degree of fragmentation of a file or a disk. The basic syntax:

```
BLOCKS fileidi <fileido> ( DATA NODISP INT nn FRAGMAP
```

*BLOCKS fileidi*

produces the following kind of display.

Blocks	Low	High	Pieces	Jmplen	Bkjmps
100	120	507	9	287	0

- Blocks - the number of data blocks in the file.
- Low, High - the extent of disk blocks within which the file resides.
- Pieces - the count of separate, contiguous stretches of data blocks.
- Jmplen - the total length of the gaps between the pieces. If there are any backward jumps in the file, this length can easily be greater than the difference between Low and High.
- Bkjmps - the number of times when following the logical, ascending, order of the file, it is necessary to go 'backwards' on the disk — from higher to lower disk block numbers.

On an empty disk, CMS tries to allocate contiguous data blocks in ascending order. A very fragmented file would have a difference between Low and High much larger than the file size; it would have many Pieces, a Jmplen several times the Low-High extent difference, and several backward jumps. For a very smooth and compact file, the Low-High difference would be almost the same as the data block count, the number of pieces would be only 1 or 2, the jump length would be 0 to 2, and there would be one or no backward jumps.

The two-line display described above may be followed by a "map" of the distribution of blocks in the file. The extent of the file between the Low and High blocks is divided into equal intervals. The count of the number of data blocks of the file which fall into each interval is displayed, 20 per line. The default interval size is 128. The resolution provided by this size may be too gross. The interval size can be changed with the INT nn option. The minimum interval size that will be used is 4. If the 'nn' specified in the INT option is not a power of 2, it will be rounded up to a power of 2. The block distribution map is not displayed if it would show only 1 or 2 intervals.

*BLOCKS fileidi fileido*

If an output FN, FT, and FM is specified, all the index blocks at all levels for the input file are written to the output file. The output is formatted as 64-byte records. The first record is the FST of the input file. The next 2 records

contain the VOL1 label of the input disk. Starting at record 4 is the root pointer block, followed by the lower level pointer blocks. Most of this data is in hex form. Recfm F files have 4-byte pointer elements, recfm V files have 12-byte pointers.

The DATA option causes as many as possible of the data blocks of the input file also to be copied into the output, following the index blocks. The BLOCKS program allocates only one, large memory buffer. If it is not sufficient to hold all of the data blocks, the ending data blocks of the input file will not appear in the output. For a 1-block file, no output at all is written unless the DATA option is used. The NODISP option suppresses the display of information at the terminal, but still allows an output file to be written.

The DIRECTOR and ALLOCMAP can be analyzed and dumped much like any other file. To process one of these two, specify the filetype of fileid to be 'DIRECTOR' or 'ALLOCMAP'. The filename in this case is ignored and can be anything generally valid. Since the allocmap is rarely more than 1 block of data, it can practically never be fragmented. The bits in the allocmap though specify the block distribution, and fragmentation, on the whole minidisk. FRAGMAP is a special option which allows a distribution map of the whole disk to be displayed. If it is specified, the DATA option should be used at the same time, and the input filetype should be ALLOCMAP. The INT option controls as described above the degree of resolution of this map. The default interval size is again 128. So, the following variation of the BLOCKS command should display a fragmentation map for all allocated blocks of a minidisk.

*BLOCKS file allocmap a (data int 64 fragmap*

Below is sample output for a perfectly defragmented file.

```
BLOCKS database file b
Blocks  Low  High  Pieces  Jmplen  Bkjmps
13047   121  15000   14     14892   1
```

File block distribution in intervals of 128 blocks of disk space.

```
128 127 128 128 128 128 128 128 128 127 128 128 128 128 128 127 128 128
128 128 128 128 128 127 128 128 128 128 128 128 127 128 128 128 128 128
128 127 128 128 128 128 128 128 128 127 128 128 128 128 128 127 128 128
128 128 128 128 128 127 128 128 128 128 128 128 127 128 128 128 128 128
128 127 128 128 128 128 128 101 0 0 0 0 0 0 0 0 0 0 0
0 0 127 128 128 128 128 128 128 128 126 128 128 128 128 128 32
```

Each count of 128 indicates that all blocks in that interval of 128 are data blocks of the file. Every 8th interval has 127 data blocks because 1 index block per 1024 data blocks is required. The 'rotating cursor' method of space allocation was adopted in VM Release 5. Although the beginning of the disk is free, this method causes the beginning of the file to be allocated starting near the end of the disk space — at the 127 after the stretch of zeros. The file is written to the end of the disk — to the partial ending interval with 32 blocks. There are 2 index blocks in the interval with count 126, the first second level index and the second first level index. The cursor wraps around to the beginning of the disk, and the file is smoothly written until it ends with the partial interval of 101. The gap of 0's is actually between the end and beginning of the file, and is unavoidable with the rotating cursor method of space allocation. The low block number used is not any lower because the file system reserves a percentage of the disk at the beginning for the directory. The 14 pieces of the file are caused only because of the small gaps of the index blocks. The one backward jump is due to the rotating cursor and is for the entire length of the disk extent. This one jump is for 15000-121 blocks, which constitutes the total jump length except for the 13 1-block gaps for the index blocks.

## FILE SYSTEM CONSISTENCY CHECKING

The BLOCKS command can also perform file system consistency checks when the VALIDATE option is given. The options described above for fragmentation analysis are ignored if VALIDATE is in effect. The remaining options described below are only significant when VALIDATE is specified. The typical command for file system checking:

```
BLOCKS * * fm (val
```

The checking can be restricted to a file or a group of files by entering a specific filename and/or filetype. Although the disk must be accessed, BLOCKS analyzes the directory and allocmap on the disk rather than the ones in memory. R/O access to the disk is sufficient for complete validation. R/W access is only necessary for the FIX option described below.

The primary kinds of validation performed:

- Every block in use by a file should be reserved in the allocation bit map, and conversely.
- A block should not be shared, i.e., used by more than one file, or possibly used more than once by the same file.
- The number of blocks indicated to be in use by the file pointers should agree with the file size shown in the directory entry, the FST.
- FST fields should be legal.
- The extra information in recfm V file pointers should be within valid limits.

Many, specific messages are displayed for a variety of detected problems. Some errors do not affect the basic integrity of the file system, such as non-zero CDF fields in an FST, or invalid date-time fields in an FST. Other errors are critical and indicate that if data or the entire disk have not been lost, they soon would be. The two most important kinds of deadly errors are (1) used but unallocated blocks, and (2) shared blocks. Occasionally, files or disks suffering these problems can be salvaged. We would recommend making a DDR backup of the broken disk before trying to fix it. Simple disk block editing can be performed with the RDBLK and WRBLK commands distributed with XCOPY. Please call Sequential Software if we could help rescue important data from the brink.

BLOCKS can fix two kinds of problems with the disk allocation bit map. One is when blocks are allocated but not used, which just decreases the amount of free space that should be available. The other problem is used, but unallocated blocks, which leaves the disk on the verge of chaos. If it discovers errors, BLOCKS will display a message telling whether it can fix the disk or not. If it is fixable, the FIX option can be specified as in the following.

```
ACC cuu fm  
BLOCKS * * fm (val fix
```

The access command is to insure that the allocmap in memory matches the one on disk. BLOCKS does not issue an access internally. There must be write access to the disk because its allocmap will be updated. The FIX option is only effective when the filename and filetype are '\* \*'.

```
BLOCKS * * fm (val map
```

The above command generates a simple map of disk block usage in the file BLOCK MAP A1. The map indicates for each disk block which file is using it, or whether it is free. Although the lrecl of the BLOCK MAP file is set to 32, it basically consists of a string of 4-byte records, one per disk block. The low order 3 bytes of each 4 bytes contains a file number. The DIRECTOR is file 1, the ALLOCMAP is file 2. The first file which appears in a LISTFILE display is file 3, the second LISTFILE item is file 4, and so on. If the command

```
BLOCKS file director fm my dir a (data
```

is issued, MY DIR A will contain a list of files which correspond in sequence one-for-one with the file numbers which appear in the map.

A zero file number in the map indicates a free block. The high order byte of the 4-bytes per block contains the level number of the block in the file system structure. A zero level number indicates a data block, 1 indicates a first level pointer block, 2 a second level pointer block, etc. A few initial disk blocks are reserved, and they are given a special map entry of X'FFFFFFE'. These are the ipl records, the disk label, and the beginning of the one-back directory.

The map file contains only hex data. To view it with XEDIT, one must enter 'V H1 32'. Editing the map can give a quick visual sense of how fragmented the files and free space are.

If BLOCKS quits with a message indicating insufficient storage, it may be possible to run with the VStorage option. BLOCKS must be able to fit the following items into memory.

- The entire disk directory (64 bytes per file).
- All pointer blocks at all levels for the largest file on the disk.
- If the MAP option is used, a map file of size (4 bytes)\*(disk size in blocks).
- Sundry smaller work areas. A certain simple buffer space is given 384K by default. This can be adjusted up or down with the VSTORAGE option. For example,

*BLOCKS \* \* fm (val vs 256K*

---

# BLOCKS Messages and Return Codes

## **BLCKS001E Disk “mode” not accessed. RC=4**

*The disk to be analyzed must be accessed.*

## **BLCKS002E Only minidisks are supported, not Shared File System directories. RC=8**

*BLOCKS does not provide any analysis of accessed SFS directories.*

## **BLCKS003E Disk “mode” is non-EDF. RC=12**

*BLOCKS does not analyze CDF disks.*

## **BLCKS004I Blocks Low High Pieces Jmplen Bkjmps’**

*The header for fragmentation information displayed on the next line. ‘Blocks’ identifies the total data block count of the file. ‘Low’ and ‘High’ give the extent range of the file in relative disk block numbers. ‘Pieces’ is the number of discontinuous stretches of data blocks in the file. ‘Jmplen’ is the sum in blocks of the lengths of the gaps between all of the discontinuous pieces. ‘Bkjmps’ is the count of occasions when the logical order of file data blocks requires a backward jump on the disk, toward a lower relative disk block number.*

## **BLCKS005I**

*This message number is assigned to the line of numbers identified by the preceding message, BLCKS004I.*

## **BLCKS006I**

*This message number is assigned to a blank spacer line.*

## **BLCKS007I File block distribution in intervals of “nn” blocks of disk space.**

*This header precedes a display of the distribution of disk block counts found during fragmentation analysis.*

## **BLCKS008I**

*This message number is assigned to a line of interval block counts displayed after fragmentation analysis.*

## **BLCKS009E Cmd format is either BLOCKS fileidi <fileido> (DATA NODISP INT nn FRAGMAP RC=16**

*The format of the BLOCKS command line was incorrect. This message is displayed along with BLCKS044E to describe the valid formats.*

## **BLCKS010E Insufficient core. RC=20**

*Enough main memory must be available to hold the entire disk directory plus all of the index blocks of the largest file on the disk. A large area is allocated which is usually sufficient. If it proves not to be, this message would be displayed. Try running the command again with the VSTORAGE option to increase memory beyond 384K. This message would also appear if there is insufficient free storage available in the virtual machine. The VSTORAGE option can also be used in this case to reduce the default allocation.*

## **BLCKS011E File “fn ft fm” not found. RC=28**

*The input file specified on the command line does not exist.*

## **BLCKS012E I/O error “nn” processing file “fn ft fm”. RC=nn**

*BLOCKS attempts to circumvent file system problems which would cause I/O errors, but this is not always possible.*

**BLCKS013E Disk origin pointer is not 4 or 5. It is “nn” (in hex). RC=32**

*At +X'10' in the disk label should be a fullword containing either 4 or 5. This is the disk block number of the beginning of the directory. No further processing is attempted when this number is invalid.*

**BLCKS014E Disk block size invalid — “nn”. RC=36**

*The value found in the disk label should be 512, 1K, 2K, or 4K. No further processing is attempted.*

**BLCKS015E {ADTUSED | ADTAMNB} field in disk label invalid — “value”. RC=36**

*A value greater than the maximum number of blocks on the disk was found in the disk label.*

**BLCKS016E File “fn ft fm”, ptr blk nbr “nn” invalid var len disp — “mm”. RC=40**

*The last word of an index block for a recfm V file should contain the displacement to the last index entry in the block. An impossible value was found for one of these displacements. This message typically could appear when the pointer to the index block points incorrectly to a data block.*

**BLCKS017E File “fn ft fm”, blk nbr “nn” invalid. RC=40**

*An index block for the named file contains the invalid block pointer which is displayed. The pointer is greater than the maximum number of blocks on the disk.*

**BLCKS018E File “fn ft fm”, blk nbr “nn”, var len index info invalid. RC=40**

*Block number “nn” is an index block of the named recfm V file. The second and third words of each of its 12-byte pointer elements should contain the high record number in the data block pointed to and the offset to the first full record in the data block.*

**BLCKS019E File “fn ft fm”, blk nbr “nn”, unallocated in bit map. RC=40**

*This is a severe error. Unless it can be fixed very soon, some or all of the data on the disk may be lost. If the disk does not already contain shared blocks, the FIX option of BLOCKS may be able to repair it.*

**BLCKS020E File “fn ft fm”, blk nbr “nn” shared. RC=40**

*Bad news. At best, a data block of one file has been clobbered. To salvage as much as possible of the disk, all files on it should be copied using XCOPY's CHECK option.*

**BLCKS021I File “fn ft fm” is sparse.**

*The file contains blocks of all binary zeroes. These blocks have not been allocated on the disk, but are indicated by index pointers to the non-existent disk block 0. This is a valid, but fairly rare circumstance prior to VM/SP6. SP6 was changed to create sparse files automatically whenever blocks of all zeroes would be written.*

**BLCKS022E File “fn ft fm” has a zero block ptr, but should not be sparse. RC=40**

*The FST indicates the file should not be sparse, i.e., there should not be any zero disk block pointers.*

**BLCKS023E File “fn ft fm”, FST indicates sparse, but no sparse blks found. RC=40**

*For a recfm F file, the block count represents less data than the lrecl times the record count. In other words, the file should be sparse, but no zero index pointers were found. The lrecl, the record count, and/or the block count in the FST are probably incorrect.*

**BLCKS024E I/O error, RDTK RC “nn” reading the first “nn” block(s) in the following list. RC=36**

*The DMSDIO return code is displayed. BLOCKS generally should avoid this error by giving valid block lists to DMSDIO. A permanent disk read error is possible, but rare. The block numbers in the DMSDIO plist are displayed in hex following this message.*

**BLCKS025E**

*This message number is assigned to the block list displayed after BLCKS024E.*

**BLCKS026E The following disk blocks are allocated but not used (hex). RC=36**

*This is a relatively innocuous error which just wastes disk space, although it may indicate a more serious problem is brewing. The excess bits in the allocmap can be turned off by means of the FIX option. A list of the particular block numbers are displayed in hex in a following list.*

**BLCKS027E “beg blk nbr” (- “end blk nbr”) (hex)**

*This message number is assigned to the block list displayed after BLCKS026E and BLCKS044E.*

**BLCKS028E Disk “fm” is R/O, cannot be fixed. RC=36**

*The FIX option was specified, but the disk cannot be updated because it is accessed R/O.*

**BLCKS029E Severe errors found. Disk will not be fixed. RC=36**

*Errors were found which could not be corrected by the FIX option. The disk is left unchanged.*

**BLCKS030E File “fn ft fm” has invalid characters in the fileid. RC=36**

*An FST was found in the disk directory which contained characters in its fileid which are normally invalid.*

**BLCKS031E File “fn ft fm”, FM number invalid. RC=36**

*Only 1-6 are valid filemode numbers.*

**BLCKS033E File “fm ft fm”, FST F-V flag invalid — “flag value”. RC=36**

*Only ‘F’ or ‘V’ is valid. Perhaps junk FST entries have crept into the disk directory.*

**BLCKS034E File “fn ft fm”, FSTPTRSZ invalid — “ptr value”. RC=36**

*It should be 4 for recfm F, or 12 for recfm V.*

**BLCKS035I File “fn ft fm”, blk count too high.**

*For a recfm F file, the maximum data block count can be determined by multiplying the record count by the Irecl. This maximum value has been exceeded for the named file. The large block count probably would not cause a file system error by itself. This count can be reduced to agree with the record count by copying the file.*

**BLCKS036E File “fn ft fm”, file origin ptr invalid — “ptr value”. RC=36**

*This disk block number should not be zero, nor greater than the total block count of the disk.*

**BLCKS037I File “fn ft fm”, FST date-time invalid.**

*By itself, this problem does not cause a file system error. An impossible value was found for yy, mm, dd, hh, mm, or ss.*

**BLCKS038E File “fn ft fm”, FSTNLVL inconsistent with file size.**

*The number of levels of index pointers can be 0 to 5. A certain number of levels is required for a certain file size.*

**BLCKS039I From disk label, “nn” blocks used, “mm” total blocks.**

*The block counts obtained directly from the disk label.*

**BLCKS040I From file ptrs, “nn” blocks used, “mm” shared, “pp” unallocated.**

*Normally the “nn” blocks used matches the count from the disk label, and the shared and unallocated counts are zero. The disk is probably unusable, or on the brink of becoming so if either of the latter two counts are non-zero. ‘Shared’ means two or more files are using the same data block. ‘Unallocated’ means that a file is using a block which is marked free to be used by new files. If a problem with unallocated blocks is caught early enough, BLOCKS can correct it with the FIX option, unless the following message appears.*

**BLCKS041I The detected problems are not corrected by the FIX option.**

*One or more severe errors were found. They cannot be repaired by a subsequent execution of BLOCKS with the FIX option.*

**BLCKS042E The following blocks are used but not allocated (hex). RC=36**

*Used but unallocated blocks jeopardize the entire disk. When new files are written, they might overlay existing files, including possibly the directory. The numbers of the blocks in question are displayed in hex in a following list.*

**BLCKS043I There are errors which should be correctable with the FIX and VALIDATE options.**

*The errors found which related to disk block allocation can be fixed by another execution of BLOCKS with the FIX and VALIDATE options. If there is any doubt about whether the incore allocation map matches the one on disk, the disk should be reaccessed prior to running with FIX. The disk should not be released, only reaccessed.*

**BLCKS044E or BLOCKS <fn,\*> <ft,\*> fm (VALidate MAP FIX VStorage <nK,NM> RC=16**

*The second line showing the basic BLOCKS command format, in case it was entered with some syntax error.*

# CMPR

Use the CMPR command to find the differences between two files. These files might typically be program sources or exec's. The format of the command:

```
CMPR fileid1 fileid2 [ fileid3 ] ( DDD CC mm nn
```

Fileid1 and fileid2 are full file identifiers for the two files to be compared. Any of the three parts of fileid2 may be specified by an equal sign (=), indicating that the corresponding part of fileid1 is to be used. If a third file identifier is given, any differences will be written to a file under this name. When fileid3 is not given, the differences are displayed at the terminal. When the output showing the file differences is typed on the screen rather than written to a file, no more than the first 80 bytes of a record is displayed. When output is to a file, the full records are always written. The amount of virtual memory required is approximately 1.5 times the filesize of the second file.

The DDD option stands for Don't Display Deletes. It suppresses the display of any records deleted or replaced. If one is not interested in these records, DDD can save a little time by avoiding a second pass through the first file. The CC mm nn option specifies that records should be distinguished based on the data in positions mm to nn inclusive. Without this option, records are compared for their whole length. 'nn' can be specified as '\*', which means to compare to the end of the record. The compare length is automatically shortened for each record if nn extends beyond the end of the record. As an example, when comparing source files, CC 1 72 might be desired to ignore sequence numbers. Two variable length records must have the same length to be considered equal, unless the compare field is given by the CC option.

The format in which differences are displayed is shown below. The numbers on the ./ lines are relative record numbers within the first file.

```
./ I nnn                the records immediately following
_____                were inserted after record nnn in
_____                file 1.
_____

./ D mmm [nnn]         record mmm or records mmm through
./ * * * DELETED RECORDS * * *      nnn were deleted.
-----                The data which was removed is shown here.
_____
_____

./ R mmm [nnn]         record mmm or records mmm through
_____                nnn were replaced by the following.
_____

./ * * * REPLACED RECORDS * * *
-----                The deleted data is shown after what was inserted.
_____
```

If no differences are discovered, the command ends with return code 0; otherwise, the return code is 8.

The MRGCHG exit provided with XCOPY can apply the changes found by the CMPR command. Therefore, if two versions of a source are compared with CMPR, the second file could be erased. It could be reconstructed by applying the difference file to the first file by a command such as the following. MRGCHG would expect the change file to be called fn DIFF, in this case — CUR DIFF.

```
XCOPY CUR FILE A ONE BACK A (RECEX MRGCHG
```

## Messages:

Want either 2 or 3 full fileids.

Insufficient core.

Error nn reading file fn ft fm.

---

# FCOMPARE

Use the FCOMPARE command to determine whether two CMS files are identical or not. The command format:

```
FCOMPARE fn1 ft1 fm1 fn2 ft2 fm2 ( BEOF
```

In the second fileid, fn2, ft2, or fm2 can be specified as "=", in which case the equal sign is interpreted the same as the corresponding part of the first fileid. Files on CDF disks are not supported. If the data blocks of the two files contain bit-for-bit the same data up to EOF, the command ends with a zero return code and no messages. The whole last block beyond end-of-file is checked if the BEOF option is given.

## Messages:

**Disk "mode" not accessed. RC=32**

**Disk "mode" is non-EDF. RC=36**

**Insufficient storage. RC=40**

*About 132K of main memory is required.*

**Command parms are two full fileids. RC=44**

*The command line was not entered properly.*

**File fn ft fm not found. RC=28**

**Error nn reading file fn ft fm. RC=nn**

**Files differ at byte displacement dddd. RC=48**

**Recfm, lrecl, or rcd ct in fsts do not match. RC=48**

**File {1 | 2} ended early. RC=48**

---

# MACCOMP

Use the MACCOMP command to compress MACLIB's. It performs the COMP function of the MACLIB command in considerably less time. The command format:

```
MACCOMP  fn [ MACLIB [[ fm ]] [ fileido ]
```

The first fileid gives the name of the MACLIB to be compressed. If the filetype is specified, it must be MACLIB. The filemode defaults to A if it is not given. If a second fileid is given, the compressed MACLIB is written under this name. When only the first fileid is specified, the command first writes the compressed MACLIB to the same disk under a temporary name. The uncompressed MACLIB is then erased, and the new file is renamed. This procedure requires that the disk containing the uncompressed MACLIB must have enough free space to contain the compressed MACLIB. Specifying a second fileid in the command line allows a different disk to be specified for the compressed file.

## Messages:

**Error nn {reading | writing} file fn ft fm. RC=nn**

**Insufficient storage. RC=8**

**Command parms are FN MACLIB FM (FN2 FT2 FM2). RC=8**

**Invalid second fileid. RC=8**

**Invalid maclib format. RC=8**

---

# RDBLK

Use the RDBLK command to display disk blocks or to display an entry in an incore disk directory. The command has two formats.

```
RDBLK filename filetype filemode  
RDBLK filemode hexblkno
```

Use the first format to display the directory entry of the specified file. Use the second format to display a certain disk block. In this latter case, the first parameter specifies the access letter of the disk from which to read. The second parameter, hexblkno, is the disk block number in hexadecimal. The file directory entry contains the block number, in hexadecimal, of the file origin at +X'28' beyond where the filename and filetype appear at the beginning. The disk block is read into memory at location X'F000'. The first 128 bytes are displayed from this location by an internal CP DISPLAY command. More of the block can be shown by entering one's own DISPLAY commands.

Messages:

**Disk not accessed.**

**File not found.**

---

## XCTIME

Use the XCTIME EXEC to compare the performance of two different commands. It displays the vtime, ttime, elapsed time, and SIO count required by each command. The arguments entered for XCTIME are passed on to the commands to be measured. In the form in which it is provided, the EXEC compares COPYFILE and XCOPY. Therefore, in the following examples,

```
XCTIME * * I = = T
XCTIME DMSSP MACLIB S OSMACRO MACLIB S OUT FILE T
```

the copy commands are timed for the two operations of copying all files from the S-disk and concatenating the two MACLIB's into one file.

XCTIME prompts for the CUU and a filemode which can be used to access a temporary disk where the test output is written. Please note that all files are erased from this disk when it is accessed by the EXEC. When the output disk is specified among the command line arguments, this same filemode letter should be given. In the examples above, "T" should be the response to the prompt for a filemode. Entering "XCTIME ?" provides a reminder of how to use the EXEC.

The routine in the EXEC named "dcmd" is called to measure one command. The calls to this routine could be altered to compare more or different commands.

---

## XCBENCH

Use the XCBENCH EXEC to produce a report comparing the performance of two or more copyfile replacement products. It automatically generates test data and runs about 40 test cases. Included in the tests are both simpler copies and more complex ones which present a variety of challenging performance problems. As this EXEC is distributed, it performs comparisons between XCOPY and COPYFILE. Comments at the beginning of the EXEC indicate how it can be easily changed to compare other and/or more commands. The results of the tests are displayed as they run and are collected at the end into a report called XCBENCH LISTING A.

XCBENCH requires two temporary disks which each can hold about 16 megabytes of data. If the command is entered without any parameters, it will define and format these disks. It tries to use device addresses 2B1 and 3B1. Particular work disks can be specified to be used for the tests. For instance, XCBENCH 251 351 would cause devices 251 and 351 to be used for the work disks. XCBENCH would format the work areas at 4K. If it is desired to test with other block sizes, the temporary disks should be defined, formatted, and specified to XCBENCH.

An FC option can be specified to cause the output of all the programs being tested to be compared against "standard" files created by IBM's COPYFILE. The FCOMPARE utility is used to perform the comparisons. A third temp disk will be automatically defined when FC is in effect. This disk can also be specified as a third CUU parameter in the XCBENCH command. The time to complete the benchmarks increases significantly with the FC option because of the extra COPYFILE executions.

Statistics are collected for Vtime, Ttime, elapsed time, and SIO's. The tests are numbered in the reports 1, 2, 3, ... . Certain tests can be selectively omitted by editing their test numbers into a string in the exec assigned to the variable 'Omits'.

# XCCOUNT

A facility is provided by which COPYFILE usage can be measured in the following categories: frequency of use, elapsed time, vtime, ttime, and sio's. This is useful to the system programmer who needs to quantify the savings that XCOPY will provide when installed as a replacement for COPYFILE. This facility is intended to be run for some appropriate time before XCOPY replaces COPYFILE. A report can be generated which summarizes the above performance criteria. This data then can be compared with XCOPY performance benchmarks to estimate the savings that XCOPY will provide.

There are three components of the XCCOUNT facility:

1- A service virtual machine which collects and saves statistics must be set up by the user. The primary program running in this machine is XCSERV, which establishes the IUCV environment and awaits IUCV/SMSG type interrupts which contain copyfile statistics from another machine. This machine can run disconnected and needs an A-disk with sufficient space to write the accounting records. The XCSERV EXEC contains the necessary statements to start up this component. To stop this machine and to flush statistics from main memory to disk issue:

```
SMSG userid SHUTDOWN
```

where *userid* is that of the service machine.

2- XCCREP is an EXEC which reads and processes the statistics collected by XCSERV. It reads file "XCSERV ACCOUNT A" and creates a report in file "XCCREP LISTING A", which can be displayed or printed as desired. This file contains the summary of COPYFILE usage, with totals and averages for elapsed time, vtime, ttime, and sio's.

3- XCCOUNT is a module which front-ends the COPYFILE command, and as such must be made available to each CMS machine in which COPYFILE usage is to be measured. A system profile should be used to activate this module by inserting the following command in it:

```
NUCXLOAD COPYFILE XCCOUNT (SYSTEM
```

XCCOUNT MODULE must be placed on a disk which is available when the system profile is executed. The XCCOUNT program must also know to which service machine to send the statistics via SMSG; the XCINST EXEC can be run to define this userid to XCCOUNT. No special classes or definitions are necessary for either the service machine or the user's virtual machine.

The overhead of running this facility should be minimal. This description is intended to give a good overview of how to set up XCCOUNT; please call Sequential Software for any additional help in using it or in interpreting the report generated by XCCREP.

---

# XCF

Use XCF to prepare a disk for use with the CMS file system. XCF is a high-performance replacement for the FORMAT command. For more than just a few cylinders, XCF runs in about one-third the time of FORMAT for CKD devices. XCF calls the standard FORMAT command to build the directory structure and then performs the bulk of the I/O itself. It is reusable and can run as a nucleus extension if desired. The command format:

```
XCF cuu mode [ nbr of cyls | nbr of blocks ] [ (options... [ ] ) ]
```

The minidisk to be formatted is at virtual address *cuu*. After formatting completes, this disk will be available under the CMS *mode* letter specified in the second parameter. Both the *cuu* and *mode* parameters are required. To format just part of a minidisk, specify either the number of cylinders or blocks as the third parameter. Omission of this parameter indicates that the entire minidisk is to be formatted.

## Options

### **Blksize { 512 | 800 | 1024 | 2048 | 4096 | 1K | 2K | 4K }**

is the number of bytes written per DASD block on the CMS minidisk. A CKD device must be used to format with an 800 byte block size (an older disk format with several limitations). Any other value indicates an extended disk format (EDF) disk. In recent releases of VM, the default block size for CKD devices has become 4K. The default for FBA devices is 1K.

### **Noerase**

For FBA devices, this specifies that the disk blocks are not cleared to binary zeroes. This is the default with XCF, as considerable performance gains are realized by not having to write every FB-512 block on the minidisk. For CKD devices, a minidisk that is accessed as the same filemode letter indicated on the command line, is assumed to be properly formatted. Only its directory structure will be re-written, when the Noerase option is specified, which acts much like the *ERASE* option of the ACCESS command.

### **Label**

Rewrites the volume label on a minidisk. No formatting is performed. Message DMSFOR605R will be issued requesting a volume label to be entered. The first six characters entered are used, or if fewer then the label is left-justified and right-padded with blanks. The QUERY DISK command can display the current volume label.

### **Recomp**

A disk must already have been formatted to use this option. The number of cylinders or blocks specified as the third parameter on the command line redefines the amount of space available to the file system. If this parameter is omitted, then for EDF disks the redefined disk size equals the maximum number of cylinders/blocks previously formatted. For 800 byte disks, all cylinders are used. A decrease in size may not be permitted since existing files may reside beyond the recompute limit. A COPY \* \* A (REP operation will generally not be sufficient to reorganize a disk so all allocated blocks are clustered at the beginning. Instead, all the data files should be moved to some intermediate area, such as T-disk, tape, or virtual reader files. Then erase the files from the disk, recompute with XCF, and re-load the files.

## TYPE

displays at the terminal the product name and release number.

## IOLIMIT n

allows the maximum number of tracks formatted per SIO to be specified. The default provided is 3. Whatever track limit may be specified, XCF will not format more than a cylinder at-a-time. The default IOLIMIT can be altered by the XCINST exec.

## FILL hh

causes the disk to be cleared with repetitions of the byte specified by the two hex digits 'hh'. The default fill character is the usual 00. FILL FF, for example, would clear the disk with bytes of X'FF'. The only blocks of the disk which are not filled in this way are the initial few, typically 6, necessary for the initial file system structure. FILL has no effect for FBA disks.

## Usage Notes

1) Except when using RECOMP or LABEL, any files on the disk are effectively erased. A prompt is issued to help prevent the accidental loss of data.

2) No read-verify after format-write check is performed. This is rarely needed for modern DASD. If the hardware is under suspicion for some reason, use the standard FORMAT command to perform the read-verify check.

## Messages

These messages are in addition to those found under the FORMAT command in the *CMS Command Reference Manual*.

### **XCFMT003E Insufficient storage. RC=3**

*XCF requires more storage than normal Format. Make more virtual storage available and try XCF again.*

### **XCFMT004E I/O error nn during partial cylinder format. RC=nn**

*An error has occurred trying to format the minidisk. nn is the return code from Diagnose 20 or A8.*

### **XCFMT005E I/O error nn during full cylinder format. RC=nn**

*An error has occurred trying to format the minidisk. nn is the return code from Diagnose 20 or A8.*

### **XCFMT008E XCF license has expired. Please call Sequential Software. RC=8**

*The trial or permanent version of the product has expired.*

### **XCFMT009I XCF Rel. nn Copyright Sequential Software, Inc.**

*Produced when the TYPE option is specified and identifies the XCF release.*

### **XCFMT010E "IBMFOR TEXT" not found on an accessed disk. RC=10**

*When XCF runs from a loaded TEXT file, it searches for CMS FORMAT by this name.*

### **XCFMT011E Error nn on LOAD of IBMFOR TEXT. RC=nn**

*The CMS LOAD command failed loading this file.*

**XCFMT012E Error while QUERYing the DASD. Response buffer too small by nn bytes. RC=nn**

*XCF when necessary uses the response from QUERY V cuu to determine the disk size. This message appears if the response buffer provided by XCF to CP through Diag 8 is not large enough. Please call Sequential for assistance.*

**XCFMT014E Invalid option "option". RC=14**

*The option displayed in the message is incorrect or has an invalid parameter.*

# ZZAP

Use ZZAP to change the contents of a CMS module file. Statements indicating the changes to be made are supplied either through a CMS file or through the console. These statements are syntax compatible with the regular CMS ZAP program. ZZAP runs faster than ZAP as it only updates the module on disk when there is no more zapping to perform. If no filename is specified, ZZAP prompts at the terminal for zap statements. The filetype defaults to "ZAP" if not entered, and the filemode defaults to "A". The command format is as follows.

```
ZZAP [ filename ] [filetype [filemode]] [( options [ ] )]
```

ZZAP can prevent most typographical errors if special hash-check codes are included following each VER and REP statement. The hash-check code will be indicated by a colon followed by two hexadecimal digits. An error message is produced and the zap terminates if the hash-check fails. A recommended way of applying zaps would be to run ZZAP with the VERO option to check the verify data and hash-check numbers, if any, before actually trying to update the module on disk.

```
ZZAP myfix (VERO
```

## Options

### HASH

will cause the zap file to be re-written with hash-check numbers on each VER and REP statement. This option is generally used by Sequential before distributing a fix.

### MAINT

suppresses any verify reject messages at the terminal. It also displays a list of zaps successfully applied. This option is used by Sequential when distributing a set of zaps constituting some maintenance level.

### VEROnly

processes just the verify statements. The module is not updated on disk. This is useful to check that all verify data is correct before applying maintenance. It can help prevent a multi-csect zap from being partially applied, when part of a zap verifies correctly and another part does not.

### UNZap

inverts the usual meaning of VERify and REPlace statements. This option can back-off a previously applied zap. When used in conjunction with VEROnly, a module can be checked to insure that a particular zap is applied. Specifically, VEROnly prevents the module from being updated and UNZap causes the REP statements to be used to verify data.

# Index

## A

access 57  
AFTER 5, 21  
APPEND 10, 18

## B

BEFORE 5, 21  
BLKSIZE 12, 28, 57  
BLOCKS 10, 42  
BYPASS 21, 30

## C

CASE 21, 22  
CDATE 21  
CHECK 12, 30  
CMPR 51  
compare 51, 52  
copyfile cmsut1 16

## D

DATA 42  
DECRYPT 20  
DESDEC 20  
DESEND 20

## E

EBCDIC 19  
ENCRYPT 19  
ERASE 19, 20  
EXEC 20  
EXECNAME 20

## F

FCOMPARE 52  
FIFO 20  
file system 45, 57  
fileid 10  
filemode 0 5, 12, 30  
FILL 10, 17, 18, 22, 58  
FIX 42  
FM0 30  
FOR 21  
FORMAT 12, 28, 57  
format 5, 57  
FRAGMAP 42  
fragmentation 10, 43  
FRLABEL 21, 22, 29  
FROM 21  
FRSTRING 22, 29

FSTEXIT 24

## H

HASH 60  
HX 19

## I

installation 7, 39, 40  
INTERVAL 42  
IOLIMIT 58

## L

LABEL 57  
LIFO 20  
LOWCASE 19  
LRECL 10, 18

## M

MACCOMP 53  
MAINT 60  
maintenance 60  
MAP 42  
messages 33, 47, 58  
minidisk move 11

## N

NEWDATE 16  
NEWER 23  
NEWFILE 16  
NOCOPY 29  
NODISP 42  
NOERASE 57  
NOEXT 24  
NOPACKWN 22, 23  
NOPROMPT 16  
NOSPARE 30  
NOSPECS 18  
NOSTR 28  
NOTRUNC 18  
NOTYPE 16  
NOUPDIRT 20

## O

OLDDATE 16  
OLDDDATE 12  
OLDER 23  
overlay 10  
OVLY 17, 18

## P

PACK 22, 23  
PACKED 23  
PACKWNG 23  
pattern matching 10

PERCENT 24  
performance 5, 10, 24, 28, 55  
PPACK 22  
PREERASE 29  
PROMPT 16

## R

r/o extension 5  
RDBLK 54  
RECEXIT 25  
RECFM 10, 18  
RECLRECL 28  
RECOMP 57  
REORG 28  
REPLACE 16, 20, 23  
REONLY 23  
Reserved minidisks 11  
REXX 5

## S

SAMEDATE 23  
SFS 12  
SINCE 21  
SINGLE 19  
SKIPREP 23  
SOFF 21, 29  
SON 29  
SPARSE 29  
SPECS 16, 17  
STACK 20  
STRINIT 28  
sysprof exec 8

## T

TO 22  
TOLABEL 22, 29  
TOSTRING 22, 29  
TRANS 16, 19  
TRUNC 18  
TYPE 16, 29, 58

## U

UNPACK 23  
UNPACKED 23  
UNZAP 60  
UPCASE 19  
UPDIRT 21

## V

VALIDATE 42  
VERONLY 60  
VM/ESA 5  
VSTORAGE 11, 24, 43

## X

XCBENCH 55  
xcbench 7  
XCCOUNT 56  
XCF 28, 57  
XCINST 58  
XCTIME 55  
xctime 7

## Y

Year 2000 5, 21

## Z

zap 60  
ZZAP 60