FLEX User's Manual

PREFACE

The purpose of this User's Guide is to provide the user of the FLEX Operating System with the information required to make effective use of the available system commands and utilities. This manual applies to FLEX 9.0 for full size and mini floppy disks. The user should keep this manual close at hand while becoming familiar with the system. It is organized to make it convenient as a quick reference guide, as well as a thorough reference manual.

TABLE OF CONTENTS

Note:
* indicates programs
supplied by GIMIX

I. INTRODUCTION

The FLEX™ Operating System is a very versatile and flexible operating system. It provides the user with a powerful set of system commands to control all disk operations directly from the user's terminal. The systems programmer will be delighted with the wide variety of disk access and file management routines available for personal use. Overall, FLEX is one of the most powerful operating systems available today.

The FLEX Operating System is comprised of three parts, the File Management System (FMS), the Disk Operating System (DOS), and the Utility Command Set (UCS). Part of the power of the overall system lies in the fact that the system can be greatly expanded by simply adding additional utility commands. The user should expect to see many more utilities available for FLEX in the future. Some of the other important features include: fully dynamic file space allocation, the automatic "removal" of defective sectors from the disk, automatic space compression and expansion on all text files, complete user environment control using the TTYSET utility command, and uniform disk wear due to the high performance dynamic space allocator.

The UCS currently contains many very useful commands. These programs reside on the system disk and are only loaded into memory when needed. This means that the set of commands can be easily extended at any time, without the necessity of replacing the entire operating system. The utilities provided with FLEX perform such tasks as the saving, loading, copying, renaming, deleting, appending, and listing of disk files. There is an extensive CATalog command for examining the disk's file directory. Several environment control commands are also provided. Overall, FLEX provides all of the necessary tools for the user's interaction with the disk.

----
* FLEX is a registered trademark of Technical Systems Consultants, Inc.

II. SYSTEM REQUIREMENTS

FLEX requires random access memory from location 0000 through location 2FFF hex (12K). Memory is also required from C000 (48K) through DFFF hex (56K), where the actual operating system resides. The system also assumes at least 2 disk drives are connected to the controller and that they are configured as drives #0 and #1. You should consult the disk drive instructions for this information. FLEX interfaces with the disk controller through a section of driver routines and with the operator console or terminal through a section of terminal I/O routines.

III. GETTING THE SYSTEM STARTED

Each FLEX system diskette contains a binary loader for loading the operating system into RAM. There needs to be some way of getting the loader off of the disk so it can do its work. This can be done by either hand entering the bootstrap loader provided with the disk system, or by using the boot provided in ROM if appropriate to FLEX.

As a specific example, suppose the system we are using has SWTPc's S-BUG installed and we wish to run FLEX. The first step is to power on all equipment and make sure the S-BUG prompt is present (>). Next insert the system diskette into drive 0 (the boot must be performed with the disk in drive 0) and close the door on the drive. Type "D" on the terminal if using a full size floppy system or "U" if a minifloppy system. The disk motors should start, and after about 2 seconds, the following should be displayed on the terminal:

```
FLEX X.X
DATE (MM,DD,YY)?

+++
```

The name FLEX identifies the operating system and the X.X will be the version number of the operating system. At this time the current date should be entered, such as 7,3,79. The FLEX prompt is the three plus signs (+++), and will always be present when the system is ready to accept an operator command. The '+++' should become a familiar sight and signifies that FLEX is ready to work for you!

IV. DISK FILES AND THEIR NAMES

All disk files are stored in the form of 'sectors' on the disk and in this version, each sector contains 256 'bytes' of information. Each byte can contain one character of text or one byte of binary machine information. A maximum of 340 user-accessible sectors will fit on a single-sided mini disk or 1140 sectors on a single-sided full size floppy. Double-sided disks would hold exactly twice that number of sectors. Double-density systems will hold more still. The user, however, need not keep count, for the system does this automatically. A file will always be at least one sector long and can have as many as the maximum number of sectors on the disk. The user should not be concerned with the actual placement of the files on the disk since this is done by the operating system. File deletion is also supported and all previously used sectors become immediately available again after a file has been deleted.

All files on the disk have a name. Names such as the following are typical:

```
PAYROLL
INVNTORY
TEST1234
APRIL-78
WKLY-PAY
```

Anytime a file is created, referenced, or deleted, its name must be used. Names can be most anything but must begin with a letter (not numbers or symbols) and be followed by at most 7 additional characters, called 'name characters'. These 'name characters' can be any combination of the letters 'A' through 'Z' or 'a' through 'z', any digit '0' through '9', or one of the two special characters, the hyphen (-) or the underscore '_', (a left arrow on some terminals).

File names must also contain an 'extension'. The file extension further defines the file and usually indicates the type of information contained therein. Examples of extensions are: TXT for text type files, BIN for machine readable binary encoded files, CMD for utility command files, and BAS for BASIC source programs. Extensions may contain up to 3 'name characters' with the first character being a letter. Most of the FLEX commands assume a default extension on the file name and the user need not be concerned with the actual extension on the file. The user may at anytime assign new extensions, overiding the default value, and treat the extension as just part of the file name. Some examples of file names with their extensions follow:

```
APPEND.CMD
LEDGER.BAS
TEST.BIN
```

Note that the extension is always separated from the name by a period '.'. The period is the name 'field separator'. It tells FLEX to treat the following characters as a new field in the name specification.

A file name can be further refined. The name and extension uniquely define a file on a particular drive, but the same name may exist on several drives simultaneously. To designate a particular drive a 'drive number' is added to the file specification. It consists of a single digit (0-3) and is separated from the name by the field separator '.'. The drive number may appear either before the name or after it (after the extension if it is given). If the drive is not specified, the system will default to either the 'system' drive or the 'working' drive. These terms will be described a little later.

Some examples of file specifications with drive numbers follow:

```
0.BASIC
MONDAY.2
1.TEST.BIN
LIST.CMD.1
```

In summary, a file specification may contain up to three fields separated by the field separator. These fields are; 'drive', 'name', and 'extension'. The rules for the file specification can be stated quite concisely using the following notation:

```
[<drive>.]<name>[.<extension>]
  or
<name>[.<extension>][.<drive>]
```

The '<>' enclose a field and do not actually appear in the specification, and the '[]' surround optional items of the specification. The following are all syntactically correct:

```
0.NAME.EXT
NAME.EXT.0
NAME.EXT
0.NAME
NAME.0
NAME
```

Note that the only required field is the actual 'name' itself and the other values will usually default to predetermined values. Studying the above examples will clarify the notation used. The same notation will occur regularly throughout the manual.

V. ENTERING COMMANDS

When FLEX is displaying '+++', the system is ready to accept a command line. A command line is usually a name followed by certain parameters depending on the command being executed. There is no 'RUN' command in FLEX. The first file name on a command line is always loaded into memory and execution is attempted. If no extension is given with the file name, 'CMD' is the default. If an extension is specified, the one entered is the one used. Some examples of commands and how they would look on the terminal follow:

```
    +++TTYSET
    +++TTYSET.CMD
    +++LOOKUP.BIN
```

The first two lines are identical to FLEX since the first would default to an extension of CMD. The third line would load the binary file 'LOOKUP.BIN' into memory and, assuming the file contained a transfer address, the program would be executed. A transfer address tells the program loader where to start the program executing after it has been loaded. If you try to load and execute a program in the above manner and no transfer address is present, the message, 'NO LINK' will be output to the terminal, where 'link' refers to the transfer address. Some other error messages which can occur are 'WHAT?' if an illegal file specification has been typed as the first part of a command line, and 'NOT THERE' if the file typed does not exist on the disk.

During the typing of a command line, the system simply accepts all characters until a 'RETURN' key is typed. Any time before typing the RETURN key, the user may use one of two special characters to correct any mistyped characters. One of these characters is the 'back space' and allows deletion of the previously typed character. Typing two back spaces will delete the previous two characters. The back space is initially defined to be a 'control H' but may be redefined by the user using the TTYSET utility command. The second special character is the line 'delete' character. Typing this character will effectively delete all of the characters which have been typed on the current line. A new prompt will be output to the terminal, but instead of the usual '+++' prompt, to show the action of the delete character, the prompt will be '???'. Any time the delete character is used, the new prompt will be '???', and signifies that the last line typed did not get entered into the computer. The delete character is initially a 'control X' but may also be redefined using TTYSET.

As mentioned earlier, the first name on a command line is always interpreted as a command. Following the command is an optional list of names and parameters, depending on the particular command being entered. The fields of a command line must be separated by either a space or a comma. The general format of a command line is:

```
     <command>[,<list of names and parameters>]
```

A comma is shown, but a space may be used. FLEX also allows several commands to be entered on one command line by use of the 'end of line' character. This character is initially a colon (':'), but may be user defined with the TTYSET utility. By ending a command with the end of line character, it is possible to follow it immediately with another command. FLEX will execute all commands on the line before returning with the '+++' prompt. An error in any of the command entries will cause the system to terminate operation of that command line and return with the prompt. Some examples of valid command lines follow:

```
     +++CAT 1
     +++CAT 1:ASN S=1
     +++LIST LIBRARY:CAT 1:CAT 0
```

As many commands may be typed in one command line as desired, but the total number of characters typed must not exceed 128. Any excess characters will be ignored by FLEX.

One last system feature to be described is the idea of 'system' and 'working' drives. As stated earlier, if a file specification does not specifically designate a drive number, it will assume a default value. This default value will either be the current 'system' drive assignment or the current 'working' drive assignment. The system drive is the default for all command names, or in other words, all file names which are typed first on a command line. Any other file name on the command line will default to the working drive. This version of FLEX also supports automatic drive searching. When in the auto search mode if no drive numbers are specified, the operating system will first search drive 0 for the file. If the file is not found, drive 1 will be searched and so on. When the system is first initialized the auto drive searching mode will be selected. At this time, all drive defaults will be to drive 0. It is sometimes convenient to assign drive 1 as the working drive in which case all file references, except commands, will automatically look on drive 1. It is then convenient to have a diskette in drive 0 with all the system utility commands on it (the 'system drive'), and a disk with the files being worked on in drive 1 (the 'working drive'). If the system drive is 0 and the working drive is 1, and the command line was:

```
     +++LIST TEXTFILE
```

FLEX would go to drive 0 for the command LIST and to drive 1 for the file TEXTFILE. The actual assignment of drives is performed by the ASN utility. See its description for details.

VI. COMMAND DESCRIPTIONS

There are two types of commands in FLEX, memory resident (those which actually are part of the operating system) and disk utility commands (those commands which reside on the disk and are part of the UCS). There are only two resident commands, GET and MON. They will be described here while the UCS is described in the following sections.

GET

The GET command is used to load a binary file into memory. It is a special purpose command and is not often used. It has the following syntax:

```
GET[,<file name list>]

where <file name list> is: <file spec>[,<file spec>] etc.
```

Again the '[]' surround optional items. 'File spec' denotes a file name as described earlier. The action of the GET command is to load the file or files specified in the list into memory for later use. If no extension is provided in the file spec, BIN is assumed, in other words, BIN is the default extension. Examples:

```
GET,TEST
GET,1.TEST,TEST2.0
```

where the first example will load the file named 'TEST.BIN' from the assigned working drive, and the second example will load TEST.BIN from drive 1 and TEST2.BIN from drive 0.

MON

MON is used to exit FLEX and return to the hardware monitor system such as S-BUG. The syntax for this command is simply MON followed by the 'RETURN' key.

NOTE: to re-enter FLEX after using the MON command, you should enter the program at location CD03 hex.

UTILITY COMMAND SET

The following pages describe all of the utility commands currently included in the UCS. You should note that the page numbers denote the first letter of the command name, as well as the number of the page for a particular command. For example, 'B.1.2' is the 2nd page of the description for the 1st utility name starting with the letter 'B'.

COMMON ERROR MESSAGES

Several error messages are common to many of the FLEX utility commands. These error messages and their meanings include the following:

NO SUCH FILE. This message indicates that a file referenced in a particular command was not found on the disk specified. Usually the wrong drive was specified (or defaulted), or a misspelling of the name was made.

ILLEGAL FILE NAME. This can happen if the name or extension did not start with a letter, or the name or extension field was too long (limited to 8 and 3 respectively). This message may also mean that the command being executed expected a file name to follow and one was not provided.

FILE EXISTS. This message will be output if you try to create a file with a name the same as one which currently exists on the same disk. Two different files with the same name are not allowed to exist on the same disk.

SYNTAX ERROR. This means that the command line just typed does not follow the rules stated for the particular command used. Refer to the individual command descriptions for syntax rules.

GENERAL SYSTEM FEATURES

Any time one of the utility commands is sending output to the terminal, it may be temporarily halted by typing the 'escape' character (see TTYSET for the definition of this character). Once the output is stopped, the user has two choices: typing the 'escape' character again or typing 'RETURN'. If the 'escape' character is typed again, the output will resume. If the 'RETURN' is typed, control will return to FLEX and the command will be terminated. All other characters are ignored while output is stopped.

APPEND

The APPEND command is used to append or concatenate two or more files, creating a new file as the result. Any type of file may be appended but it only makes sense to append files of the same type in most cases. If appending binary files which have transfer addresses associated with them, the transfer address of the last file of the list will be the effective transfer address of the resultant file. All of the original files will be left intact.

DESCRIPTION

The general syntax for the APPEND command is as follows:

```
     APPEND,<file spec>[,<file list>],<file spec>
```

where <file list> can be an optional list of the specifications. The last name specified should not exist on the disk since this will be the name of the resultant file. If the last file name given does exist on the disk, the question "MAY THE EXISTING FILE BE DELETED?" will be displayed. A Y response will delete the current file and cause the APPEND operation to be completed. A N response will terminate the APPEND operation. All other files specified must exist since they are the ones to be appended together. If only 2 file names are given, the first file will be copied to the second file. The extension default is TXT unless a different extension is used on the FIRST FILE SPECIFIED, in which case that extension becomes the default for the rest of the command line. Some examples will show its use:

```
     APPEND,CHAPTER1,CHAPTER2,CHAPTER3,BOOK
     APPEND,FILE1,1.FILE2.BAK,GOODFILE
```

The first line would create a file on the working drive called 'BOOK.TXT' which would contain the files 'CHAPTER1.TXT', CHAPTER2.TXT', and 'CHAPTER3.TXT' in that order. The second example would append 'FILE2.BAK' from drive 1 to FILE1.TXT from the working drive and put the result in a file called 'GOODFILE.TXT' on the working drive. The file GOODFILE defaults to the extension of TXT since it is the default extension. Again, after the use of the APPEND command, all of the original files will be intact, exactly as they were before the APPEND operation.

ASN

The ASN command is used for assigning the 'system' drive and the 'working' drive or to select automatic drive searching. The system drive is used by FLEX as the default for command names or, in general, the first name on a command line. The working drive is used by FLEX as the default on all other file specifications within a command line. Upon initialization, FLEX assigns drive #0 as both the system and working drive. An example will show how the system defaults to these values:

```
     APPEND,FILE1,FILE2,FILE3
```

If the system drive is assigned to be #0 and the working drive is assigned to drive #1, the above example will perform the following operation: get the APPEND command from drive #0 (the system drive), then append FILE2 from drive #1 (the working drive) to FILE1 from drive #1 and put the result in FILE3 on drive #1. As can be seen, the system drive was the default for APPEND where the working drive was the default for all other file specs listed.

Automatic drive searching causes FLEX to automatically scan the ready drives for the file specified. Hardware limitations prevent the mini floppy versions from searching for "ready" drives. For this reason, FLEX has been setup to ALWAYS assume drive 0 and 1 are ready. Thus if a mini floppy version of FLEX attempts to search a drive which does not have a disk loaded, it will hang up until a disk is inserted and the door closed. Alternatively, the system reset could be hit and a warm start executed (a jump to address $CD03). The full size floppy version CAN detect a ready condition and will not check drives which are out of the ready state during automatic drive searching.

Automatic drive searching causes FLEX to first check drive #0 for the file specified. If not there (or if not ready in the full size version), FLEX skips to drive #1. If the file is not found on drive #1 in the mini floppy version, FLEX gives up and a file not found error results. In the full size version FLEX continues to search on drives #2 and #3 before reporting an error.

DESCRIPTION

The general syntax for the ASN command is as follows:

```
     ASN[,W=<drive>][,S=<drive>]
```

where <drive> is a single digit drive number or the letter A. If just ASN is typed followed by a 'RETURN', no values will be changed, but the system will output a message which tells the current assignments of the system and working drives, for example:

```
     +++ASN
     THE SYSTEM DRIVE IS #0
     THE WORKING DRIVE IS #0
```

Some examples of using the ASN command are:

```
ASN,W=1
ASN,S=1,W=0
```

where the first line would set the working drive to 1 and leave the system drive assigned to its previous value. The second example sets the system drive to 1 and the working drive to 0. Careful use of drive assignments can allow the operator to avoid the use of drive numbers on file specifications most of the time!

If auto drive searching is desired, then the letter A for automatic, should be used in place of the drive number.

```
Example:
ASN W=A
ASN S=A, W=1
ASN S=A, W=A
```

The BUILD command is provided for those desiring to create small text files quickly (such as STARTUP files, see STARTUP) or not wishing to use the optionally available FLEX Text Editing System. The main purpose for BUILD is to generate short text files for use by either the EXEC command or the STARTUP facility provided in FLEX.

DESCRIPTION

The general syntax of the BUILD command is:

```
BUILD,<file spec>
```

where <file spec> is the name of the file you wish to be created. The default extension for the spec is TXT and the drive defaults to the working drive. If the output file already exists the question "MAY THE EXISTING FILE BE DELETED?" will be displayed. A Y response will delete the existing file and build a new file while a N response will terminate the BUILD command.

After you are in the 'BUILD' mode, the terminal will respond with an equals sign ('=') as the prompt character. This is similar to the Text Editing System's prompt for text input. To enter your text, simply type on the terminal the desired characters, keeping in mind that once the 'RETURN' is typed, the line is in the file and can not be changed. Any time before the 'RETURN' is typed, the backspace character may be used as well as the line delete character. If the delete character is used, the prompt will be '???' instead of the equals sign to show that the last line was deleted and not entered into the file. It should be noted that only printable characters (not control characters) may be entered into text files using the BUILD command.

To exit the BUILD mode, it is necessary to type a pound sign ('#') immediately following the prompt, then type 'RETURN'. The file will be finished and control returned back to FLEX where the three plus signs should again be output to the terminal. This exiting is similar to that of the Text Editing System.

The CATalog command is used to display the FLEX disk file names in the directory on each disk. The user may display selected files on one or multiple drives if desired.

DESCRIPTION

The general syntax of the CAT command is:

```
     CAT[,<drive list>][,<match list>]
```

where <drive list> can be one or more drive numbers seperated by commas, and <match list> is a set of name and extension characters to be matched against names in the directory. For example, if only file names which started with the characters 'VE' were to be cataloged, then VE would be in the match list. If only files whose extensions were 'TXT' were to be cataloged, then .TXT should appear in the match list. A few specific examples will help clarify the syntax:

```
     +++CAT
     +++CAT,1,A.T,DR
     +++CAT,PR
     +++CAT,0,1
     +++CAT,0,1,.CMD,.SYS
```

The first example will catalog all file names on the working drive or on all drives if auto drive searching is selected. The second example will catalog only those files on drive 1 whose names begin with 'A' and whose extensions begin with 'T', and also all files on drive 1 whose names start with 'DR'. The next example will catalog all files on the working drive (or on all drive if auto drive searching is selected) whose names start with 'PR'. The next line causes all files on both drive 0 and drive 1 to be cataloged. Finally, the last example will catalog the files on drive 0 and 1 whose extensions are CMD or SYS.

During the catalog operation, before each drive's files are displayed, a header message stating the drive number is output to the terminal. The name of the diskette as entered during the NEWDISK operation will also be displayed. The actual directory entries are listed in the following form:

```
     NAME.EXTENSION    SIZE PROTECTION CODE
```

where size is the number of sectors that file occupies on the disk. If more than one set of matching characters was specified on the command line, each set of names will be grouped according to the characters they match. For example, if all .TXT and .CMD files were cataloged, the TXT types would be listed together, followed by the CMD types.

In summary, if the CAT command is not parameterized, then all files on the assigned working drive will be displayed. If a working drive is not assigned (auto drive searching mode) the CAT command will display files on all on line drives. If it is parameterized by only a drive number, then all files on that drive will be displayed. If the CAT command is parameterized by only an extension, then only files with that extension will be displayed. If only the name is used, then only files which start with that name will be displayed. If the CAT command is parameterized by only name and extension, then only files of that root name and root extension (on the working drive) will be displayed. Learn to use the CAT command and all of its features and your work with the disk will become a little easier.

The current protection code options that can be displayed are as follows:

D	File is delete protected (delete or rename prohibited)
W	File is write protected (delete, rename and write prohibited)
(blank)	No special protection

COPY (-TSC,-GMX)†

The COPY command is used for making copies of files on a disk. Individual files may be copied, groups of name-similar files may be copied, or entire disks may be copied. The COPY command is a very versatile utility. When files are copied onto a newly formatted disk, they are stored as contiguous groups of sectors, resulting in minimum access times. This can be a substantial improvement over an old disk on which the files are highly fragmented due to frequent rewriting.

DESCRIPTION

The general syntax of the COPY command has three forms:

```
a. COPY,<file spec>,<file spec>
b. COPY,<file spec>,<drive>
c. COPY,<drive>,<drive>[,<match,list>]
```

where <match list> is the same as that described in the CAT command and all rules apply to matching names and extensions. When files are copied, if the destination disk has a file with the same name as the file being copied, the file name and the message "FILE EXISTS - DELETE ORIGINAL?" will be displayed on the console. Typing "Y" will cause the file on the destination disk to be deleted and the file on the source disk will be copied to the destination disk. Typing "N" will direct FLEX not to copy the file in question.

The first type of COPY allows copying of a single file into another. The output file may be on a different drive, but if it is on the same drive then the file names must be different. It is always necessary to specify the input file's extension, but the output file's extension will default to that of the input file if none is specified. Example:

```
+++COPY,0.TEST.TXT,1.TEST25
```

This command line would cause the file TEST.TXT on drive 0 to be copied to a file named TEST25.TXT on drive 1. Note that the destination file's extension defaulted to TXT, the same as the input file.

The second type of COPY allows copying a file from one drive to another with the file name unchanged. Example:

```
+++COPY,0.LIST.CMD,1
```

Here the file named LIST.CMD on drive 0 would be copied to drive 1. It is again necessary to specify the file's extension in the file specification. This form of the command is more convenient than the first if the copied file is to have the same name.

The final form of the COPY command is the most versatile and the most powerful. With this form,it is possible to copy all the files on one drive to another drive, or only those files which match one of the patterns in the match list. Examples:

```
     +++COPY,0,1
     +++COPY,1,0,.CMD,.SYS
     +++COPY,0,1,A,B,CA.T
```

The first example would copy all the files on drive 0 to drive 1. The second example would copy all CMD and SYS files on drive 1 to drive 0. The third example would copy from drive 0 to drive 1 all files beginning with the letter A or the letter B, or beginning with the letters CA and with an extension beginning with the letter T. This form of the COPY command is the most versatile because it allows a set of files to be extracted from a disk. The file name is always preserved with this form. During execution, the name of each file copied is displayed on the console along with the drive to which it is copied.

The match list is processed as follows: for each partial file specification in the list, all the entries in the catalog of the source disk are tested and those that match are copied. Then the whole catalog is scanned again for matches to the next specification in the list. Thus all the files which match a given specification will be grouped together in the catalog of the output disk. If a file matches more than one specification in the list, then COPY will try to copy it as many times as it matches. Example:

```
     +++COPY,1,2,ABC,.TXT
```

would copy the file ABC.TXT twice. The second time would generate the "FILE EXISTS - DELETE ORIGINAL?" prompt.

† Two versions of COPY are supplied with GIMIX FLEX 4.x. Except for the manner in which the file creation date is handled they are functionally identical. Use the RENAME utility to change the name of the preferred version to COPY.CMD.

COPY-TSC creates its output file through the normal FMS file creation function. Therefore the creation date of the output file is the current system date. This is the standard version of COPY normally supplied with FLEX.

COPY-GMX has been modified by GIMIX so that the creation date of the output file will be the same as that of the input file. For all files except random-access files, this date is the last date on which the file's contents were altered, and is often very useful to know. This version of COPY allows all copies of a file with the same contents to have the same date.

CHECKSUM

The CHECKSUM command performs a 32 bit checksum on an entire disk. The program reads every sector on the disk and totals them together. This can be used to verify disk copies, check disk validity, etc.

DESCRIPTION

The general synatax of the CHECKSUM command is:

```
     CHECKSUM[,dn]
```

Where 'dn' is an optional drive number. If no drive is specified CHECKSUM will use the work drive. If the work drive is set to 'ALL' an error message is printed. Some examples follow:

```
     +++CHECKSUM
     +++CHECKSUM,2
```

The first example will generate a CHECKSUM of the disk in the current work drive, assuming the work drive in not set to 'ALL'. The second example will generate a CHECKSUM of the disk in drive 2. The output of CHECKSUM will look like:

```
     CHECKSUM:  0002AB02
```

CHECKSUM can generate the following error messages:

```
     ILLEGAL DRIVE NUMBER
```

Legal drive numbers are 0, 1, 2, or 3. A drive number must be specified if the work drive is set to ALL.

```
     INVALID DISK FORMAT
```

The disk uses a non-standard format or the SYSTEM INFORMATION RECORD sector may be damaged.

CMPBIN

The CMPBIN command is used to compare the contents of two binary files and list the differences. CMPBIN is a useful tool for sorting out mislabeled or long-forgotten binary files, for tracking changes in programs, and for identifying current versions.

DESCRIPTION

The general syntax of the CMPBIN command is:

```
CMPBIN,<file spec>,<file spec>
```

This will cause the two files to be read as FLEX binary files and compared. The default extension is BIN. The files are read as binary records, in the format described on page 45 of the FLEX Advanced Programmer's Guide. A binary record consists of a load address, a byte count, and bytes of data to be stored in memory. The data bytes from the first file are compared to the data bytes from the second file. The current load address for each file is also compared. If either is different, the address and data byte from each file is printed. Example:

```
+++CMPBIN,A.BIN,AOLD.BIN
   FILE A         FILE B
ADDRESS  BYTE   BYTE  ADDRESS
  0209    A5     A7    0209
  020A    56     29    020A
  03E4    C6     4D    03E4
```

CMPBIN is a very simple-minded program, and works best only when the two files load starting at the same address. If the files differ by one file having code inserted or removed, then mismatches will be found from the point where bytes were added or removed to the end of the file. If one file is longer than the other, the extra bytes will all be mismatches, with the shorter file's contents listed as "0106   00". If the data bytes are the same, but the files were assembled to load at different addresses, then every byte will be a mismatch, but only on the addresses, which is easily seen.

DATE

The DATE command is used to display or change an internal FLEX date register. This date register may be used by future programs and FLEX utilities.

DESCRIPTION

The general syntax of the DATE command is:

```
     DATE[,<month,day,year>]
```

where 'month' is the numerical month, 'day' is the numerical day and 'year' is the last two digits of the year.

```
     +++DATE 5,2,79  Sets the date register to May 2, 1979
```

Typing DATE followed by a carriage return will return the last entered date.

```
     Example:
             +++DATE
             May 2, 1979
```

DELETE

The DELETE command is used to delete a file from the disk. Its name will be removed from the directory and its sector space will be returned to the free space on the disk.

DESCRIPTION

The general syntax of the DELETE command is:

```
DELETE,<file spec>[,<file list>]
```

where <file list> can be an optional list of file specifications. It is necessary to include the extension on each file specified. As the DELETE command is executing it will prompt you with:

```
DELETE "FILE NAME"?
```

The entire file specification will be displayed, including the drive number. If you decide the file should be deleted, type 'Y'; otherwise, any other response will cause that file to remain on the disk. If a 'Y' was typed, the message 'ARE YOU SURE?' will be displayed on the terminal. If you are absolutely sure you want the file deleted from the disk, type another 'Y' and it will be gone. Any other character will leave the file intact. ONCE A FILE HAS BEEN DELETED, THERE IS NO WAY TO GET IT BACK! Be absolutely sure you have the right file before answering the prompt questions with Y's. Once the file is deleted, the space it had occupied on the disk is returned back to the list of free space for future use by other files. Few examples follow:

```
+++DELETE,MATHPACK.BIN
+++DELETE,1.TEST.TXT,0.AUGUST.TXT
```

The first example will DELETE the file named MATHPACK.BIN from the working drive. If auto drive searching is selected, the file will be deleted from the first drive it is found on. The second line will DELETE the file TEST.TXT from drive 1, and AUGUST.TXT from drive 0.

There are several restrictions on the DELETE command. First, a file that is delete or write protected may not be deleted without first removing the protection. Also a file which is currently in the print queue (see the PRINT command) can not be deleted using the DELETE command.

DCOPY

The DCOPY command is used to copy from one disk to another all files which were created on or after a given date. This permits convenient backup of only those files which are new.

DESCRIPTION

The general syntax of the DCOPY command is:

```
DCOPY,<drive>,<drive>,[<month>],[<day>],[<year>][,R]
```

where the first drive number is the drive to be copied from, the second drive number is the drive to be copied to. R is an option to replace existing files on the destination disk with the files being copied. <month>, <day>, and <year> indicate a date; only files created on or after this date will be copied. If any part of the the date is left out, DCOPY defaults to the value in the corresponding FLEX date register. For example

```
DCOPY,2,1,6,1,R
```

would copy from drive 2 to drive 1 all files created after June 1 of the current year, replacing existing copies on drive 1.

```
DCOPY,0,3,,,81
```

would copy from drive 0 to drive 3 all files created after today's date in 1981 which were not already on the disk in drive 3.

```
DCOPY,1,0,R
```

would copy from drive 1 to drive 0 all files which were created today, replacing existing copies of these files on drive 0.

DCOPY logs each file copied on the console with the message

```
n.filnam.ext TO DRIVE #n
```

These messages may be redirected to the printer or to a file with the P and O commands to provide a record of operations.

If the R option is selected, files on the destination disk with the same names as files to be copied will be deleted, and replaced by the copied files, unless the date of the destination disk file is more recent than that of the source disk file. In that case, DCOPY will print

```
DEST FILE IS NEWER - NOT COPIED
```

and go on to the next file. This prevents the user from "backing up" an out-of-date file onto the current file.

NOTE: the date of a random-access file is the day it was created. FLEX does not change this date when the file is accessed. The GIMIX-supplied UPDATE utility command can be used to update the date of a random-access file.

DIR

The DIR command is used to display the contents of a FLEX disk directory. It is similar to the CAT command, but displays all directory information associated with each file. The user may display selected files on multiple disks, and may include or exclude catalog-protected files. DIR reports the total number of files, the number of sectors in use, size of the largest file, and free sectors on each disk. NOTE: this program is a GIMIX product, and not related to any other programs of the same name.

DESCRIPTION

The general syntax of the DIR command is:

```
     DIR[,+P][,<drive list>][,<match list>]
```

where +P is the protected-files option, <drive list> is a list of up to 4 drives to be scanned, and <match list> is one or more partial file names.

If "+P" is included in the command line immediately after "DIR", then the DIR command will process all files on the disk, including those which have been catalog-protected with the PROT command. If the option is omitted, protected files are excluded from the listing but are included in the printed total at the end of the listing.

<drive list> can be one or more drive numbers separated by commas or spaces. For each drive specified, a header is displayed, directory entries matching the match list are displayed, and totals are printed. If the drive list is omitted, DIR defaults to the working drive.

<match list> consists of one or more sets of name and/or extension characters which are to be matched against file names in the directory. The sets must be separated by commas or spaces, and must be in the form [<partial name>][.<partial extension>]. The match function compares the letters in the set to the beginning of each file name (and extension), and those files which match all the letters in the set are listed. For example:

```
     +++DIR HARDF .BAK A.C TEXTFILE.TXT
```

will list all files on the working disk with names beginning with "HARDF", then all files with the extension ".BAK", then all files with names beginning with "A" and extensions begining with "C", and finally the file "TEXTFILE.TXT". A file named HARDF19.BAK would be listed twice, as a "HARDF" file, and as a ".BAK" file. For convenience, all lower case letters in the match list are translated to upper case before the comparison, so "l.dat" and "L.DAT" are equivalent, and both will match "LIST.DAT". If the match list is omitted, DIR lists all files on the drive.

For each drive DIR prints a header giving the disk name and volume number entered when the disk was formatted, and the date it was formatted. Then it scans the directory once for each item in the match list, listing all files which match. For each file, the following information is displayed: directory entry number, name and extension, an "R" if it is random-access, the disk addresses of its beginning and ending sectors, its size in sectors, the date it was last written to, and a 4-character field indicating which protections are set for it (Write, Read, Delete, or Catalog). Example:

```
+++DIR +P,3,A.CMD,LETT,ERR

Directory of drive 3
Disk name = VIRTDISK.GMX #1  Disk created 28-Mar-84

File #  Name     Type  R  Begin  End    Size  Date       Prot

   2    ASMB     .CMD     01-04  07-02   47  28-Mar-84  W.D.

  13    LETTER   .BAK     1B-01  1B-01    1  28-Mar-84  .R.C

  14    ERRORS   .SYS  R  1B-02  1B-04    3  28-Mar-84  ....


  Listed: Files=    5 :  Sectors=   51 :  Largest=   47
  Total:  Files=   35 :  Sectors=  291 :  Largest=   75 : Free=245
```

Two set of totals are printed at the bottom. The first line is for the listed files only, while the second is for the whole disk, including protected and non-matching files. If multiple disks are specified, separate totals are printed for each disk.

EXEC

The EXECute command is used to process a text file as a list of commands, just as if they had been typed from the keyboard. This is a very powerful feature of FLEX for it allows very complex procedures to be built up as a command file. When it is desirable to run this procedure, it is only necessary to type EXEC followed by the name of the command file. Essentially all EXEC does is to replace the FLEX keyboard entry routine with a routine which reads a line from the command file each time the keyboard routine would have been called. The FLEX utilities have no idea that the line of input is coming from a file instead of the terminal.

DESCRIPTION

The general syntax of the EX command is:

```
     EXEC,<file spec>
```

where <file spec> is the name of the command file. The default extension is TXT. An example will give some ideas on how EXEC can be used. One set of commands which might be performed quite often is the set to make a new system diskette on drive 1 (see NEWDISK). Normally it is necessary to use NEWDISK and then copy all .CMD and all .SYS files to the new disk. Finally the LINK must be performed. Rather than having to type this set of commands each time it was desired to produce a new system diskette, we could create a command file called MAKEDISK.TXT which contained the necessary commands. The BUILD utility should be used to create this file. The creation of this file might go as follows:

```
     +++BUILD,MAKEDISK
       =NEWDISK,1
       =COPY,0,1,.CMD,.OV,.LOW,.SYS
       =LINK,1.FLEX
       =#
     +++
```

The first line of the example tells FLEX we wish to BUILD a file called MAKEDISK (with the default extension of .TXT). Next, the three necessary command lines are typed in just as they would be typed into FLEX. The COPY command will copy all files with CMD, OV, LOW, and SYS extensions from drive 0 to drive 1. Finally the LINK will be performed. Now when we want to create a system disk we only need to type the following:

```
     +++EXEC,MAKEDISK
```

We are assuming here that MAKEDISK resides on the same disk which contains the system commands. EXEC can also be used to execute the STARTUP file (see STARTUP).

There are many applications for the EXEC command. The one shown is certainly useful but experience and imagination will lead you to other useful applications.

IMPORTANT NOTE: The EXEC utility is loaded into the very upper end of user memory. This is done by first loading EXEC into the utility file space, then calculating the proper starting address so that it will reside right up against the end of the user memory space. Next EXEC is moved to that location and a new end of memory is set to just below EXEC. When the EXEC file is finished, if the user has not further changed the memory end location, EXEC will reset it to the original value.

EXTEND

The EXTEND command is used to extend the directory space of a disk by adding sectors taken from the chain of free sectors.

DESCRIPTION

The general syntax of the COMMAND command is:

```
     EXTEND[,<drive>,<sectors>]
```

where <drive> is the number of a drive, and <sectors> is the number of sectors to be added to the directory space of the disk in that drive. If the parameters are omitted, EXTEND defaults to the current working disk and 10 sectors. Each added sector holds 10 additional directory entries. The number of sectors added must be at least 1 and no more then 255. The free chain is reduced by the number of sectors added to the directory.

EXTEND was created for users who may want to put large numbers of files on a disk. If so many files are put on a disk that the original directory space is filled up, FLEX extends the directory by adding sectors from the free chain, one at a time. This will cause the directory to become fragmented, and all disk operations requiring directory searches will become much slower. EXTEND avoids this by adding the additional sectors in a single block. If this is done before any files have been written to the disk, then the added space will be adjacent to the original directory, thus keeping the directory as compact as possible. Therefore, if the user expects to need extra directory space, EXTEND should be used immediately after the disk is formatted.

The amount of directory space on a disk is determined by the FORMAT command (for floppies) or the HARDFxxx command (for hard disks). This table gives the number of sectors allocated for each type of disk by the formatting program, and the number of directory entries available.

		sectors	entries
5 1/4" floppy	single-sided	6	60
	double-sided	16	160
8" floppy	single-sided	11	110
	double-sided	26	260
6 MB removable Winchester		60	600
19 MB Winchester		188	1880

If you expect to have more files on a disk than can be entered in the space listed above, then you should extend the directory before making any use of the disk.

Examples:

```
     +++EXTEND,2,8
```

adds 8 sectors to the directory of the disk in drive 2, making room for 80 addditional files.

```
     +++EXTEND
```

adds 10 sectors to the directory of the working disk.

EXTEND has some special restrictions: the number of sectors added must be less than the number left in the free chain. If no parameters are included on the command line (for default operation), then the working drive number may not be "ALL". If the drive number is given, then the number of sectors must also be given.

FREEMAP

The FREEMAP command is used to check the list of available sectors (free chain) on a FLEX formatted disk (floppy or hard) to determine the amount of fragmentation that exists.

DESCRIPTION

The general syntax of the FREEMAP command is:

```
     FREEMAP,<drive>
```

FREEMAP then scans all the sectors in the free chain of the disk in the designated drive, and lists on the console all the groups of continuous sectors found. The total number of such groups (called segments) is displayed at the end. By examining this list, the user can determine the degree of fragmentation of the disk, and decide whether to run UNSNARL on it, or copy the files on it to a new disk. Example:

```
     FREEMAP,1
     READING FREE CHAIN
     0908-1012
     0706-0708
     1120-1306
     2208-2209
     220C-220C

     SEGMENTS MAPPED:     5
```

The segment count is printed in decimal. The output from this program can be routed to the printer or to a file with the P or O commands.

See the UNSNARL command for more information.

The I command allows a utility to obtain input characters from a disk file rather than the terminal.

DESCRIPTION

The general syntax of the I command is:

```
     I,<file spec>,<command>
```

where <file spec> is the name of the file containing the characters to be used as input and <command> is the FLEX utility command that will be executed and that will receive that input from <file spec>. The default extension on <file spec> is .TXT.

For example, say that on a startup you always wanted the file DATA.DAT deleted from the disk without having to answer the "ARE YOU SURE?" questions. This could be done in the following manner:

```
     +++BUILD,YES
     =YY
     =#
```

The first Y will answer the "DELETE 0.DATA.DAT?" question while the second Y will answer the "ARE YOU SURE?" question.

```
     +++BUILD,STARTUP
     =I,YES,DELETE,DATA.DAT
     =#
```

Upon booting the disk, FLEX will execute the STARTUP file and perform the following operation: delete the file DATA.DAT receiving all answers to any questions from the input file YES.TXT rather than from the terminal.

See the description of the STARTUP command for more information on STARTUP.

The JUMP command is provided for convenience. It is used to start execution of a program already stored in computer RAM memory.

DESCRIPTION

The general syntax of the JUMP command is:

```
     JUMP,<hex address>
```

where <hex address> is a 1 to 4 digit hex number representing the address where program execution should begin. The primary reason for using JUMP is if there is a long program in memory already and you do not wish to load it off of the disk again. Some time can be saved but you must be sure the program really exists before JUMPing to it!

As an example, suppose we had a BASIC interpreter in memory and it had a 'warm start' address of 103 hex. To start its execution from FLEX we type the following:

```
     +++JUMP,103
```

The BASIC interpreter would then be executed. Again, remember that you must be absolutely sure the program you are JUMPing to is actually present in memory.

The LINK command is used to tell the bootstrap loader where the FLEX operating system file resides on the disk. This is necessary each time a system disk is created using NEWDISK. The NEWDISK utility should be consulted for complete details on the use of LINK.

DESCRIPTION

The general syntax of the LINK command is:

```
     LINK,<file spec>
```

where <file spec> is usually FLEX. The default extension is SYS. Some examples of the use of LINK follow:

```
     +++LINK,FLEX
     +++LINK,1.FLEX
```

The first line will LINK FLEX.SYS on the working drive, while the second example will LINK FLEX.SYS on drive 1. For more advanced details of the LINK utility, consult the "Advanced Programmers Guide".

LIST

The LIST command is used to LIST the contents of text or BASIC files on the terminal. It is often desirable to examine a files without having to use an editor or other such program. The LIST utility allows examining entire files, or selected lines of the file. Line numbers may also be optionally printed with each line.

DESCRIPTION

The general syntax of the LIST command is:

```
     LIST,<file spec>[,<line range>][,+(options)]
```

where the <file spec> designates the file to be LISTed (with a default extension of TXT),and <line range> is the first and last line number of the file which you wish to be displayed. All lines are output if no range specification is given. The LIST command supports two additional options. If a +N option is given, line numbers will be displayed with the listed file. If a +P option is given, the output will be formatted in pages and LIST will prompt for "TITLE" at which time a title for the output may be entered. The TITLE may be up to 40 characters long. This feature is useful for obtaining output on a printer for documentation purposes (see P command). Each page will consist of the title, date, page number, 54 lines of output and a hex 0C formfeed character. Entering a +NP will select both options. A few examples will clarify the syntax used:

```
     +++LIST,RECEIPTS
     +++LIST,CHAPTER1,30-200,+NP
     +++LIST,LETTER,100
```

The first example will list the file named 'RECEIPTS.TXT' without line numbers. All lines will be output unless the 'escape character' is used as described in the Utility Command Set introduction. The second example will LIST the 30th line through the 200th line of the file named 'CHAPTER1.TXT' on the terminal. The hyphen ('-') is required as the range number separator. Line numbering and page formatting will be output because of the '+NP' option. The last example shows a special feature of the range specification. If only one number is stated, it will be interpretted as the first line to be displayed. All lines following that line will also be LISTed. The last example will LIST the lines from line 100 to the end of the file. No line numbers will be output since the 'N' was omitted.

The NAME utility enables the user to change the name, extension, volume number and date in the system information sector of a disk.

DESCRIPTION

The general synatax of the NAME command is:

```
     NAME[,dn]
```

Where 'dn' is an optional drive number. If no drive is specified NAME will use the work drive. If the work drive is set to 'ALL' an error message is printed. Some examples follow:

```
     +++NAME
     +++NAME,2
```

The first example will change the information on the disk in the work drive, assuming that the work drive is not set to all. The second example will change the information on the disk in drive #2.

NAME prints the current disk name, extension, volume number and date and then prompts for the new name. The new name and extension should be entered, followed by a carriage return. Entering only a carriage return will retain the old name. NAME then prompts for the new volume number. The new volume number should be entered , followed by a carriage return. Entering only a carriage return will retain the original volume number. After the new name and volume number have been entered, NAME prompts:

```
     CHANGE DATE ('Y' OR 'N')?
```

Entering 'Y' changes the date on the disk to the "current date", Entering 'N' retains the old date.

NAME can generate the following error message:

```
     ILLEGAL DRIVE NUMBER
```

Legal drive numbers are 0, 1, 2, and 3. A drive number must be specified if the work drive is set to 'ALL'.

NOTE: If NAME is used in a command line with multipe commands, it must be the last command on the line.

The N utility enables the user to automatically answer "N" (no) to "Y or N" prompts from other utilities. The N utility is especially useful when writing EXEC files.

DESCRIPTION

The general synatax of the N command is:

```
     N,<command string>
```

Where <command string> is a valid command line to be executed. If N is used in a line with multiple commands, using the end of line character, it only affects the command immediately following it. For example:

```
     +++N,COPY,0,1
```

Will copy, from drive #0 to drive #1, only those files that do not already exist on drive #1, automatically answering "N" (no) to any "DELETE ORIGINAL?" and "ARE YOU SURE?" prompts that occur because of duplicate files on the two disks.

The O (not zero) command can be used to route all displayed output from a utility to an output file instead of the terminal. The function of O is similar to P (the printer command) except that output is stored in a file rather than being printed on the terminal or printer. Other TSC software may support this utility. Check the supplied software instructions for more details.

DESCRIPTION

The general syntax of the O command is:

```
     O,<file spec>,<command>
```

where <command> can be any standard utility command line and <file spec> is the name of the desired output file. The default extension on <file spec> is .OUT. If O is used with multiple commands per line (using the 'end of line' character ':') it will only have affect on the command it immediately precedes. Some examples will clarify its use.

```
     +++O,CAT,CAT
                    writes a listing of the current disk directory into
                    a file called CAT.OUT

     +++O,BAS,ASMB,BASIC.TXT
                    writes the assembled source listing of the text
                    source file 'BASIC.TXT' into a file called 'BAS.OUT'
                    when using the assembler
```

The P command is very special and unlike any others currently in the UCS. P is the system print routine and will allow the output of any command to be routed to the printer. This is very useful for getting printed copies of the CATalog or used with the LIST command will allow the printing of FLEX text files.

DESCRIPTION

The general syntax of the P command is:

```
P,<command>
```

where <command> can be any standard utility command line. If P is used with multiple commands per line (using the 'end of line' character), it will only have affect on the command it immediately preceeds. Some examples will clarify its use:

```
+++P,CAT
+++P,LIST,MONDAY:CAT,1
```

The first example would print a CATalog of the directory of the working drive on the printer. The second example will print a LISTing of the text file MONDAY.TXT and then display on the terminal a CATalog of drive 1 (this assumes the 'end of line' character is a ':'). Note how the P did not cause the 'CAT,1' to go to the printer. Consult the 'Advanced Programmer's Guide' for details concerning adaption of the P command to various printers.

The P command tries to load a file named PRINT.SYS from the same disk which P itself was retrieved. The PRINT.SYS file which is supplied with the system diskette contains the necessary routines to operate a SWTPC PR 40 printer connected through a parallel interface on PORT 7 of the computer. If you wish to use a different printer configuration, consult the 'Advanced Programmer's Guide' for details on writing your own printer driver routines to replace the PRINT.SYS file. The PR 40 drivers, however, are compatible with many other parallel interfaced printers presently on the market.

The PROT command is used to change a protection code associated with each file. When a file is first saved, it has no protection associated with it thereby allowing the user to write to, rename, or delete the file. Delete or write protection can be added to a file by using the PROT command.

DESCRIPTION

The general syntax of the PROT command is:

```
     PROT,<file spec>[,(option list)]
```

where the <file spec> designates the file to be protected and (option list) is any combination of the following options.

D   A 'D' will delete protect a file. A delete protected file cannot be affected by using the DELETE or RENAME Commands, or by the delete functions of SAVE, APPEND, etc.

W   A 'W' will write protect a file. A write protected file cannot be deleted, renamed or have any additional information written to it. Therefore a write protected file is automatically delete protected as well.

C   A 'C' will Catalog protect a file. Any files with a C protection code will function as before but will not be displayed when a CAT command is issued.

X   An 'X' will remove all protection options on a specific file.

Examples:

+++PROT CAT.CMD,XW	Remove any previous protection on the CAT.CMD Utility and write protect it.
+++PROT CAT.CMD,X	Remove all protection from the CAT.CMD utility.
+++PROT INFO.SYS,C	Prohibit INFO.SYS from being displayed in a catalog listing.

RENAME

The RENAME command is used to give an existing file a new name in the directory. It is useful for changing the actual name as well as changing the extension type.

DESCRIPTION

The general syntax of the RENAME command is:

```
     RENAME,<file spec 1>,<file spec 2>
```

where <file spec 1> is the name of the file you wish to RENAME and <file spec 2> is the new name you are assigning to it. The default extension for file spec 1 is TXT and the default drive is the working drive. If no extension is given on <file spec 2>, it defaults to that of <file spec 1>. No drive is requird on the second file name, and if one is given it is ignored. Some examples follow:

```
     +++RENAME,TEST1.BIN,TEST2
     +++RENAME,1.LETTER,REPLY
     +++RENAME,0.FIND.BIN,FIND.CMD
```

The first example will RENAME TEST1.BIN to TEST2.BIN. The next example RENAMEs the file LETTER.TXT on drive 1 to REPLY.TXT. The last line would cause the file FIND.BIN on drive 0 to be renamed FIND.CMD. This is useful for making binary files created by an assembler into command files (changing the extension from BIN to CMD). If you try to give a file a name which already exists in the directory, the message:

```
     FILE EXISTS
```

will be displayed on the terminal. Keep in mind that RENAME only changes the file's name and in no way changes the actual file's contents.

One last note of interest. Since utility commands are just like any other file, it is possible to rename them also. If you would prefer some of the command names to be shorter, or different all together, simply use RENAME and assign them the names you desire.

SAVE

The SAVE command is used for saving a section of memory on the disk. Its primary use is for saving programs which have been loaded into memory from tape or by hand.

DESCRIPTION

The general syntax of the SAVE command is:

```
     SAVE,<file spec>,<begin adr>,<end adr>[,<transfer adr>]
```

where <file spec> is the name to be assigned to the file. The default extension is BIN and the default drive is the working drive. The address fields define the beginning and ending addresses of the section of memory to be written on the disk. The addresses should be expressed as hex numbers. The optional <transfer address> would be included if the program is to be loaded and executed by FLEX. This address tells FLEX where execution should begin. Some examples will clarify the use of SAVE:

```
     +++SAVE,DATA,100,1FF
     +++SAVE,1.GAME,0,1680,100
```

The first line would SAVE the memory locations 100 to 1FF hex on the disk in a file called DATA.BIN. The file would be put on the working drive and no transfer address would be assigned. The second example would cause the contents of memory locations 0 through 1680 to be SAVEd on the disk in file GAME.BIN on drive 1. Since a transfer address of 100 was specified as a parameter, typing 'GAME.BIN' in response to the FLEX prompt after saving would cause the file to be loaded back into memory and execution started at location 100.

If an attempt is made to save a program under a file name that already exists, the prompt "MAY THE EXISTING FILE BE DELETED?" will be displayed. A Y response will replace the file with the new data to be saved while a N response will terminate the save operation.

Sometimes it is desirable to save noncontiguous segments of memory. To do this it would be necessary to first SAVE each segment as a separate file and then use the APPEND command to combine them into one file. If the final file is to have a transfer address, you should assign it to one of the segments as it is being saved. After the APPEND operation, the final file will retain that transfer address.

SAVE.LOW

There is another form of the SAVE command resident in the UCS. It is called SAVE.LOW and loads in a lower section of memory than the standard SAVE command. Its use is for saving programs in the Utility Command Space where SAVE.CMD is loaded. Those interested in creating their own utility commands should consult the 'Advanced Programmer's Guide' for further details.

STARTUP

STARTUP is not a utility command but is a feature of FLEX. It is often desirable to have the operating system do some special action or actions upon initialization of the system (during the bootstrap loading process). As an example, the user may always want to use BASIC immediately following the boot process. STARTUP will allow for this without the necessity of calling the BASIC interpreter each time.

DESCRIPTION

FLEX always checks the disk's directory immediately following the system initialization for a file called STARTUP.TXT. If none is found, the three plus sign prompt is output and the system is ready to accept user's commands. If a STARTUP file is present, it is read and interpreted as a single command line and the appropriate actions are performed. As an example, suppose we wanted FLEX to execute BASIC each time the system was booted. First it is necessary to create the STARTUP file:

```
+++BUILD,STARTUP
  =BASIC
  =#
+++
```

The above procedure using the BUILD command will create the desired file. Note that the file consisted of one line (which is all FLEX reads from the STARTUP file anyway). This line will tell FLEX to load and execute BASIC. Now each time this disk is used to boot the operating system, BASIC will also be loaded and run. Note that this example assumes two things. First, the disk must contain FLEX.SYS and must have been LINKed in order for the boot to work properly. Second, it is assumed that a file called BASIC.CMD actually exists on the disk.

Another example of the use of STARTUP is to set system environment paramters such as TTYSET parameters or the assigning of a system and working drive. If the STARTUP command consisted of the following line:

```
TTYSET,DP=16,WD=60:ASN,W=1:ASN:CAT,0
```

each time the system was booted the following actions would occur. First, TTYSET would set the 'depth' to 16 and the 'width' to 60. Next, assuming the 'end of line' character is the ':', the ASN command would assign the working drive to drive 1. Next ASN would display the assigned system and working drives on the terminal. Finally, a CATalog of the files on drive 0 would be displayed. For details of the actions of the individual commands, refer to their descriptions elsewhere in this manual.

As it stands, it looks as if the STARTUP feature is limited to the execution of a single command line. This is true but there is a way around the restriction, the EXEC command. If a longer list of operations is desired than will fit on one line, simply create a command file containing all of the commands desired. Then create the STARTUP file placing the single line:

```
EXEC,<file name>
```

where <file name> would be replaced by the name assigned to the command file created. A little imagination and experience will show many uses for the STARTUP feature.

By directing STARTUP to a file that does not have a return to DOS command it is possible to lockout access to DOS. You can correct the problem by hitting the RESET button and beginning execution at address $CD03. The STARTUP file may then be deleted and if desired, modified. Directing execution to CD03, the DOS warm start address, bypasses the DOS STARTUP function.

TTYSET

The TTYSET utility command is provided so the user may control the characteristics of the terminal. With this command, the action of the terminal on input and the display format on output may be controlled.

DESCRIPTION

The general syntax of the TTYSET command is:

```
     TTYSET[,<parameter list>]
```

where <parameter list> is a list of 2 letter parameter names, each followed by an equals sign ('='), and then by the value being assigned. Each parameter should be separated by a comma or a space. If no parameters are given, the values of all of the TTYSET parameters will be displayed on the terminal.

The default number base for numerical values is the base most appropriate to the parameter. In the descriptions that follow, 'hh' is used for parameters whose default base is hex; 'dd' is used for those whose default base is decimal. Values which should be expressed in hex are displayed in the TTYSET parameter listing preceded by a '$'. Some examples follow:

```
     +++TTYSET
     +++TTYSET,DP=16,WD=63
     +++TTYSET,BS=8,ES=3
```

The first example simply lists the current values of all TTYSET parameters on the terminal. The next line sets the depth 'DP' to 16 lines and the terminal width, 'WD' to 63 columns. The last example sets the backspace character to the value of hex 8, and the escape character to hex 3.

The following fully describes all of the TTYSET parameters available to the user. Their initial values are defined, as well as any special characteristics they may possess.

BS=hh   BackSpace character

This sets the 'backspace' charcter to the character having the ASCII hex value of hh. This character is initially a 'control H' (hex 08), but may be defined to any ASCII character. The action of the backspace character is to delete the last character typed from the terminal. If two backspace characters are typed, the last two characters will be deleted, etc. Setting BS=0 will disable the backspace feature.

BE=hh   Backspace Echo character

This defines the character to be sent to the terminal after a 'backspace' character is received. The character printed will have the ASCII hex value of hh. This character is initially set to a null but can be set to any ASCII character.

The BE command also has a very special use that will be of interest to some terminal owners, such as SWTPC CT-64.

If a hex 08 is specified as the echo character, FLEX will output a space (20) then another 08. This feature is very uesful for terminals which decode a hex 08 as a cursor left but which do not erase characters as the cursor is moved.

Example: Say that you mis-typed the word cat as shown below:

```
+++CAY
```

typing in one CTRL-H (hex 08) would position the cursor on top of the Y and delete the Y from the DOS input buffer. FLEX would then send out a space ($20) to erase the Y and another 08 (cursor left) to re-position the cursor.

DL=hh   DeLete character

This sets the 'delete current line' character to the hex value hh. This character is initially a 'control X' (hex 18). The action of the delete character is to 'erase' the current input line before it is accepted into the computer for execution. Setting DL=0 will disable the line delete feature.

EL=hh   End of Line character

This character is the one used by FLEX to separate multiple commands on one input line. It is initially set to a colon (':'), a hex value of 3A. Setting this character to 0 will disable the multiple command per line capability of FLEX. The parameter 'EL=hh' will set the end of line character to the character having the ASCII hex value of hh. This character must be set to a printable character (control characters not allowed).

DP=dd   DePth count

This parameter specifies that a page consists of dd (decimal) physical lines of output. A page may be considered to be the number of lines between the fold if using fan folded paper on a hard copy terminal, or a page may be defined to be the number of lines which can be displayed at any one time on a CRT type terminal. Setting DP=0 will disable the paging (this is the initial value). See EJ and PS below for more details of depth.

WD=dd   WiDth

The WD parameter specifies the (decimal) number of characters to be displayed on a physical line at the terminal (the number of columns). Lines of text longer than the value of width will be 'folded' at every multiple of WD characters. For example, if WD is 50 and a line of 125 characters is to be displayed, the first 50 characters are displayed on a physical line at the terminal, the next 50 characters are displayed on the next physical line, and the last 25 characters are displayed on the third physical line. If WD is set to 0, the width feature will be disabled, and any number of characters will be permitted on a physical line.

NL=dd   NuLl count

This parameter sets the (decimal) number of non-printing (Null) 'pad' characters to be sent to the terminal at the end of each line. These pad characters are used so the terminal carriage has enough time to return to the left margin before the next printable characters are sent. The initial value is 4. Users using CRT type terminals may want to set NL=0 since no pad characters are usually required on this type of terminal.

TB=hh   TaB character

The tab character is not used by FLEX but some of the utilities may require one (such as the Text Editing System). This parameter will set the tab character to the character having the ASCII hex value hh. This character should be a printable character.

EJ=dd   EJect count

This parameter is used to specify the (decimal) number of 'eject lines' to be sent to the terminal at the bottom of each page. If Pause is 'on', the 'eject sequence' is sent to the terminal after the pause is terminated. If the value dd is zero (which it is by default), no 'eject lines' are issued. An eject line is simply a blank line (line feed) sent to the terminal. This feature is especially useful for terminals with fan fold paper to skip over the fold (see Depth). It may also be useful for certain CRT terminals to be able to erase the previous screen contents at the end of each page.

PS=Y or PS=N   PauSe control

This parameter enables (PS=Y) or disables (PS=N) the end-of-page pause feature. If Pause is on and depth is set to some nonzero value, the output display is automatically suspended at the end of each page. The output may be restarted by typing the 'escape' character (see ES description). If pause is disabled, there will be no end-of-page pausing. This feature is useful for those using high-speed CRT terminals to suspend output long enough to read the page of text.

ES=hh   EScape character

The character whose ASCII hex value is hh is defined to be the 'escape character'. Its initial value is $1B, the ASCII ESC character. The escape character is used to stop output from being displayed, and once it is stopped, restart it again. It is also used to restart output after Pause has stopped it. As an example, suppose you are LISTing a long text file on the terminal and you wish to temporarily halt the output. Typing the 'escape character' will do this (this feature is not supported on computers using a Control Port for terminal communications). At this time (output halted), typing another 'escape character' will resume output, while typing a RETURN key will cause control to return to FLEX and the three plus sign prompt will be output to the terminal. It should be noted that line output stopping always happens at the end of a line.

The UPDATE utility enables the user to change the date in a file's directory entry to the "current date".

DESCRIPTION

The general synatax of the UPDATE command is:

```
UPDATE,<filespec>
```

Where <filespec> is the name of the file for which the date is to be changed. If the file extension is not specified, UPDATE defaults to an extension of .TXT . The file's directory entry is changed to reflect the current date. UPDATE does not alter the contents of the file itself.

UNSNARL

The UNSNARL command is used to reorganize the free chain on a FLEX disk (the list of all unused sectors). This helps reduce fragmenting of files on the disk, and improves access times.

DESCRIPTION

The general syntax of the UNSNARL command is:

```
     UNSNARL,<work drive>,<backup drive>
```

where <work drive> is the number of the drive with the disk to be reorganized, and <backup drive> is the number of the drive where the FREEMAP.TMP file can be stored. This must be different from the work drive and defaults to drive 0. UNSNARL will then read all the sectors in the free chain of the indicated disk, and make a list of all the segments in the chain.

A segment is a sector or group of sectors linked in logical order. On a newly formatted disk there is only one segment, which has all the sectors on the disk. As files are created and deleted, this segment is broken into smaller and smaller segments, and the links among them become more and more random. This results in files being stored in fragments scattered over the disk, and increases access times, especially for random-access files.

UNSNARL scans the free chain and creates a list of all the segments in the free chain. For insurance this list is saved on the backup disk as the FREEMAP.TMP file. Then UNSNARL sorts the list of segments in ascending order of disk address. Next the sector link in the last sector of each segment is pointed to the first sector of the next segment. This frequently causes several small segments to be merged into one large segment. Finally the pointers to the start and end of the free chain in the System Information Record are corrected. At each step, UNSNARL displays a descriptive message on the console. Example:

```
     +++UNSNARL,1,0
     READING FREE CHAIN
     FREE CHAIN READ, NOW SAVING MAP
     MAP SAVED, NOW SORTING EXTENT LIST
     LIST SORTED, NOW RELINKING FREE CHAIN
     RELINK DONE, DELETING MAP FILE

     +++
```

The FREEMAP.TMP file is created as insurance against power glitches or other interruptions. If UNSNARL is interrupted while it is relinking the free chain, the free chain will be left in a confused state, and creating or deleting files on that disk would be impossible. FREEMAP.TMP makes it possible for UNSNARL to pick up where it left off. The UNSN1 command is the same as UNSNARL, except that instead of reading the free chain to create the list of segments, it reads the list from the FREEMAP.TMP file. Once UNSNARL or UNSN1 successfully completes the relinking, the FREEMAP.TMP file is deleted.

UNSNARL has no effect whatever on files, so a fragmented file will remain fragmented until it is rewritten onto an unfragmented segment of free chain. Thus to keep fragmentation to a minimum UNSNARL should be used periodically, in order to clean up the fragmentation left by formerly fragmented files now using segments from the reorganized free chain.

UNSNARL only reduces file fragmentation, it does not eliminate it. The only way to eliminate fragmentation entirely is to format a new disk and perform a sequential copy of all files to the blank disk. This method is practical for floppies, but not for high capacity hard disks such as the GIMIX Winchester Disk Subsystems. Therefore GIMIX has developed the UNSNARL command as a way for users of our hard disk systems to avoid the degradation of system performance caused by severe file fragmentation.

NOTE: the degree of fragmentation in the free chain, and therefore the need for running UNSNARL, can be determined with the FREEMAP command.

VERIFY

The VERIFY command is used to set the File Management System's write verify mode. If VERIFY is on, every sector which is written to the disk is read back from the disk for verification (to make sure there are no errors in any sectors). With VERIFY off, no verification is performed.

DESCRIPTION

The general syntax of the VERIFY command is:

```
VERIFY[,ON]
   or
VERIFY[,OFF]
```

where ON or OFF sets the VERIFY mode accordingly. If VERIFY is typed without any parameters, the current status of VERIFY will be displayed on the terminal. Example:

```
+++VERIFY,ON
+++VERIFY
```

The first example sets the VERIFY mode to ON. The second line would display the current status (ON or OFF) of the VERIFY mode. VERIFY causes slower write times, but it is recommended that it be left on for your protection.

VERSION

The VERSION utility is used to display the version number of a utility command. If problems or updates ever occur in any of the utilities, they may be replaced with updated versions. The VERSION command will allow you to determine which version of a particular utility you have.

DESCRIPTION

The general syntax of the VERSION command is:

```
     VERSION,<file spec>
```

where <file spec> is the name of the utility you wish to check. The default extension is CMD and the drive defaults to the working drive. As an example:

```
     +++VERSION,0.CAT
```

would display the version number of the CAT command (from drive 0) on the terminal.

XOUT

XOUT is a special form of the delete command which deletes all files having the extension .OUT.

DESCRIPTION The general syntax of XOUT is:

```
XOUT[,<drive spec>]
```

where <drive spec> is the desired drive number. If no drive is specified all, .OUT files on the working drive will be deleted and if auto drive searching is enabled, all .OUT files on drives 1 and 2 will be deleted. XOUT will not delete any files which are delete protected or which are currently in the print queue.

Example:

```
+++XOUT
+++XOUT 1
```

The Y utility enables the user to automatically answer "Y" (yes) to
"Y or N" prompts from other utilities. The Y utility is especially useful when writing EXEC files.

DESCRIPTION

The general synatax of the Y command is:

```
     Y,<command string>
```

Where <command string> is a valid command line to be executed. If Y is used in a line with multiple commands, using the end of line character, it only affects the command immediately following it. Some examples follow:

```
     +++Y,COPY,0,1
     +++Y,DELETE,TESTFILE.CMD
```

The first example will copy all files from drive #0 to drive #1, automatically answering "Y" (yes) to any "DELETE ORIGINAL?" and "ARE YOU SURE?" prompts that occur because of duplicate files on the two disks. The second example will delete the specified file, automatically answering "Y" (yes) to the "DELETE <file spec.>?" and "ARE YOU SURE?" prompts. Use caution when using the Y utility, especially when files are being deleted, since it bypasses the normal protection against unintentionally deleting the wrong file.

GENERAL SYSTEM INFORMATION

I. DISK CAPACITY

Each sector of a FLEX disk contains 252 characters or bytes of user data (4 bytes of each 256 byte sector are used by the system). Thus a single-sided mini disk has 340 sectors or 85,680 characters or bytes of user information. A single-sided full size disk has 1140 sectors or 287,280 bytes of user data. Double-sided disks would contain exactly twice these amounts.

II. WRITE PROTECT

Floppy disks can usually be physically write protected to prevent FLEX from performing a write operation. Any attempt to write to such a disk will cause an error message to be issued. It is good practice to write protect disks which have important files on them.

A mini disk can be write protected by placing a piece of opaque tape over the small rectangular cutout on the edge of the disk. Full size floppys are just the opposite. In order to write protect a full size disk, you must remove the tape from the cutout. In other words, the notch must be exposed to write protect the disk. Some full size disks do not have this cutout and therefore cannot be write protected.

III. THE 'RESET' BUTTON

The RESET button on the front panel of your computer should NEVER BE PRESSED DURING A DISK OPERATION. There should never be a need to 'reset' the machine while in FLEX. If the machine is 'reset' and the system is writing data on the disk, it is possible that the entire disk will become damaged. Again, never press 'reset' while the disk is operating! Refer to the 'escape' character in TTYSET for ways of stopping FLEX.

IV. NOTES ON THE P COMMAND

The P command tries to load a printer driver file named PRINT.SYS from the same disk which P itself was retrieved. For the requirements of this file and on writing your own custom PRINT.SYS file, see the section on such later in this manual or consult the 'Advanced Programmer's Guide'.

V. ACCESSING DRIVES NOT CONTAINING A DISKETTE

If an attempt is made to access a minifloppy not containing a diskette, the system will hang up attempting to read until a disk is inserted and the door closed. Alternatively, you could reset the machine and begin execution at the warm start location $CD03.

VI. SYSTEM ERROR NUMBERS

Any time that FLEX detects an error during an operation, an appropriate error message will be displayed on the terminal. FLEX internally translates a derived error number into a plain language statement using a look-up table called ERROR.SYS. If you have forgotten to copy this .SYS file onto a disk that you are using, FLEX will report a corresponding number as shown below:

```
    DISK ERROR  #xx
```

where 'xx' is a decimal error number. The table below is a list of these numbers and what error they represent.

ERROR #	MEANING
1	ILLEGAL FMA FUNCTION CODE ENCOUNTERED
2	THE REQUESTED FILE IS IN USE
3	THE FILE SPECIFIED ALREADY EXISTS
4	THE SPECIFIED FILE COULD NOT BE FOUND
5	SYSTEM DIRECTORY ERROR-REBOOT SYSTEM
6	THE SYSTEM DIRECTORY IS FULL
7	ALL AVAILABLE DISK SPACE HAS BEEN USED
8	READ PAST END OF FILE
9	DISK FILE READ ERROR
10	DISK FILE WRITE ERROR
11	THE FILE OR DISK IS WRITE PROTECTED
12	THE FILE IS PROTECTED-FILE NOT DELETED
13	ILLEGAL FILE CONTROL BLOCK SPECIFIED
14	ILLEGAL DISK ADDRESS ENCOUNTERED
15	AN ILLEGAL DRIVE NUMBER WAS SPECIFIED
16	DRIVE NOT READY
17	THE FILE IS PROTECTED-ACCESS DENIED
18	SYSTEM FILE STATUS ERROR
19	FMS DATA INDEX RANGE ERROR
20	FMS INACTIVE-REBOOT SYSTEM
21	ILLEGAL FILE SPECIFICATION
22	SYSTEM FILE CLOSE ERROR
23	SECTOR MAP OVERFLOW-DISK TOO SEGMENTED
24	NON-EXISTENT RECORD NUMBER SPECIFIED
25	RECORD NUMBER MATCH ERROR-FILE DAMAGED
26	COMMAND SYNTAX ERROR-RE-TYPE COMMAND
27	THAT COMMAND IS NOT ALLOWED WHILE PRINTING
28	WRONG HARDWARE CONFIGURATION

For more details concerning the meanings of these error messages, consult the 'Advanced Programmer's Guide'.

VII. SYSTEM MEMORY MAP

The following is a brief list of the RAM space required by the FLEX Operating System. All address are in hex.

0000 - BFFF	User RAM *Note: Some of this space is used by NEWDISK, COPY and other utilities.
C000 - DFFF	Disk Operating System
C07F	System stack
C100 - C6FF	Utility command space
CD00	FLEX cold start entry address
CD03	FLEX warm start entry address

For a more detailed memory map, consult the 'Advanced Programmer's Guide'.

VIII. FLEX OPERATING SYSTEM INPUT/OUTPUT SUBROUTINES

In order for the FLEX I/O functions to operate properly, all user program character input/output subroutines should be vectored thru the FLEX operating system rather than the computer's monitor. Below is a list of FLEX's I/O subroutines and a brief description of each. All given addresses are in hexadecimal.

GETCHR at $CD15

This subroutine is functionally equivalent to S-BUG's character input routine. This routine will look for one character from the control terminal (I/O port #1) and store it in the A accumulator. Once called, the input routine will loop within itself until a character has been input. Anytime input is desired, the call JSR GETCHR or JSR $CD15 should be used.

GETCHR automatically sets the 8th bit to 0 and does not check for parity. A call to this subroutine affects the processor's registers as follows:

ACC. A	loaded with the character input from the terminal
B,X,Y,U	not affected

PUTCHR at $CD18

This subroutine is used to output one character from the computer to the control port (I/O port #1). It is functionally equivalant to the output character routine in S-BUG.

To use PUTCHR, the character to be output should be placed in the A accumulator in its ASCII form. For example, to output the letter 'A' on the control terminal, the following program should be used:

```
LDA    #$41
JSR    $CD18
```

The processor's registers are affected as follows:

ACC. A	changed internally
B,X,Y,U	not affected

PSTRNG at $CD1E

PSTRNG is a subroutine used to output a string of text on the control terminal. When address $CD1E is called, a carriage return and line feed will automatically be generated and data output will begin at the location pointed to by the index register. Output will continue until a hex 04 is seen. The same rules for using the ESCAPE and RETURN keys for stopping output apply as described earlier.

The accumulator and register status after using PSTRNG are as follows:

ACC. A	Changed during the operation

ACC. B	Unchanged
X	Contains the memory location of the last character read from the string (usually the 04 unless stopped by the ESC key)
Y,U	Unchanged

NOTE: The ability of using backspace and line delete characters is a function of your user program and not of the FLEX I/O routines described above.

For additional information consult the 'Advanced Programmer's Manual'.

STAT at $CD4E
This routine is used to determine the "status" of the input device. That is, to see if a character has been typed on the input terminal keyboard. Its function is to check for characters such as the ESCAPE key in FLEX which allows breaking of the output. This routine returns an EQual condition if no character was hit and a Not-Equal condition if a character was hit. No registers, except for the condition codes, may be altered.

COMMAND SUMMARY

```
APPEND,<file spec>[,<file list>],<file spec>
    Default extension: .TXT
    Description page: A.1

ASN[,W=<drive>][,S=<drive>]
    Description page: A.2

BUILD,<file spec>
    Default extension: .TXT
    Description page: B.1

CAT[,<drive list>][,<match list>]
    Description page: C.1

COPY,<file spec>,<file spec>
COPY,<file spec>,<drive>
COPY,<drive>,<drive>[,<match list>]
    Descripyion page: C.2

CHECKSUM[,<drive spec>]
    Description page: C.4

CMPBIN,<file spec>,<file spec>
    Description page: C.5

DATE[,<mm,dd,yy>]
    Description page: D.1

DELETE,<file spec>[,<file list>]
    Description page: D.2

DCOPY,<drive>,<drive>,[<month>],[<day>],[<year>][,R]
    Description page: D.3

DIR[,+P][,<drive list>][,<match list>]
    Description page: D.4

EXEC,<file spec>
    Default extension: .TXT
    Description page: E.1

EXTEND[,<drive spec>,<parameter>]
    Description page: E.2

FREEMAP,<drive>
    Description page: F.3

GET,<file spec>[,<file list>]
    Default extension: .BIN
    Description page: G.1

I,<file spec>,<command>
    Default extension: .TXT
    Description page: I.1

JUMP,<hex address>
    Description page: J.1
```

```
LINK,<file spec>
    Default extension: .SYS
    Description page: L.1

LIST,<file spec>[,<line range>][,N]
    Default extension: .TXT
    Description page: L.2

MON
    Description page: 1.7

NAME[,<drive spec>]
    Description page: N.1

N,<command string>
    Description page: N.2

O,<file spec>,<command>
    Default extension: .OUT
    Description page: O.1

P,<command>
    Description page: P.1

PROT,<file spec>[,(options)]
    Description page: P.3

RENAME,<file spec>,<file spec 2>
    Default extension: .TXT
    Description page: R.1

SAVE,<file spec>,<begin adr>,<end adr>[,<transfer adr>]
    Default extension: .BIN
    Description page: S.1

STARTUP
    Description page: S.2

TTYSET[,<parameter list>]
    Description page: T.1

UPDATE,<file spec>
    Default extension: .TXT
    Description page: U.1

UNSNARL,<work drive>,<backup drive>
    Description page: U.2

VERIFY[,<ON or OFF>]
    Description page: V.1

VERSION,<file spec>
    Default extension: .CMD
    Description page: V.2

XOUT[,<drive spec>]
    Description page: X.1

Y,<command string>
    Description page: Y.2
```