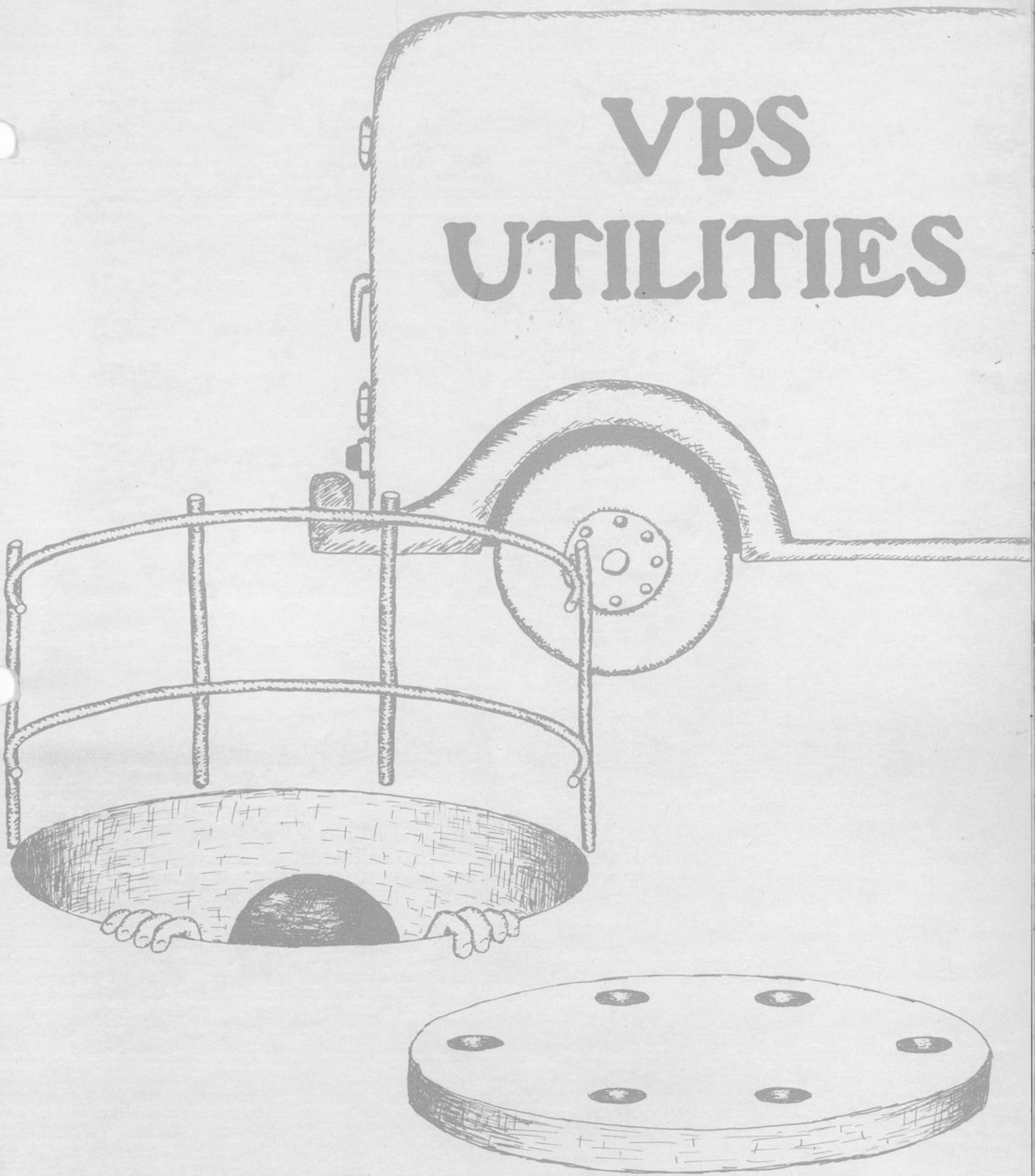
VPS UTILITIES



Boston University Computing Center

Preface

This is the third in a series of four manuals describing the features of VPS™ (the other three being - The VPS Handbook, VPS Guide to Programming Languages, and VPS System Facilities for Assembler Programmers). The VPS Handbook, which is referenced at various times in this manual, describes in detail the commands and modes of operation of the VPS system. We suggest users become familiar with the information contained in the Handbook before attempting to read this publication.

This is a general purpose manual, in that the commands and programs described should be useful to a large segment of the VPS user population.

This manual was produced entirely on VPS using the VPS Editor (/EDIT) and Script (/SCRIPT), a text preparation program. Both utilities are fully documented in Chapters 1 and 2.

We would like to acknowledge the following people who wrote some of the utilities documented in this manual: Don Feinberg (/SORT), Joe Dempty and Lew Nathan (FILELIST), and Charles Brown (the Linkage Editor and Loader).

John H. Porter
Marian G. Moore

May, 1980

Contents

Introduction

One thing all computer systems have in common is that they exist to organize and manipulate information (i.e. data). Some computer systems are highly specialized in that they are built to handle specific types of information - such as process control systems, bank teller systems, and video games. VPS, on the other hand, is a general purpose computing system which is designed to allow users to create, maintain, and process sets of information (commonly known as files) in a simple and efficient manner.

This manual describes a number of general utility programs which may be used on VPS to maintain or manipulate data files. This is by no means a complete description of all the utilities which exist on VPS; rather it is a collection of the most general, useful, and widely used programs supported by the Computing Center for information handling.

Organization of this Manual

This manual is organized into three sections by manner of invocation of the utility. The first section describes utility programs which are initiated using a VPS command (e.g. the general purpose file editor is entered using the /EDIT command). The second section describes utilities which exist as load modules and are initiated using the /LOAD statement (see the VPS Handbook for more information on /LOAD). The third section describes utility programs which are invoked using the /EXEC command (once again, see the VPS Handbook).

Below is a table showing the utilities documented here, where they are located, and a short description of their function.

Utility	Page	Function
/EDIT	3	Create and modify library or work files.
/SCRIPT	93	Format text.
/SORT	167	Sort library files.
Copy	169	Copy sequential files.
Linkage Editor	175	Create and change load modules.
Loader	191	Load object code into memory and execute it.
Compare	199	Compare two library files.
Filelist	201	List library files on the printer.
OS SORT/MERGE	203	Sort or merge library, tape, or work files.
Trcase	207	Upper/lower case translation.

Chapter 1: /EDIT - the File Editor

How to Read this Chapter

This description of the editor is intended to be reasonably concise and to avoid covering the same ground too many times. As a result, various examples of commands appear in the text even though, in most cases, these commands have not been previously described. While this will probably present no hardship to the reader already familiar with other editors, it is recommended that those new to editors regard reading this chapter as an iterative process. The steps suggested are:

1) Read all the sections down to "Logical Expressions" to get the general flavor of the editor, not worrying about the examples or other oddities.

2) Look over the command table on pages 18 and 19 and the descriptions of any commands which appear to be interesting.

3) Reread the material read in step (1), but this time worry about everything. (Reading the remaining material in this section is recommended but may be postponed until you are more comfortable with the editor.)

4) Check out the command table and command descriptions again, but this time get on a terminal and do some actual editing.

Note that a subset of the editor's functions is presented in a more tutorial fashion in the VPS Primer.

The /EDIT Command

The /EDIT command is used to run the VPS file editor. The editor is a general purpose text manipulation program used to create and modify library files (or work files, see page 15, below). Detailed explanations of library files and work files may be found in the VPS Handbook.

The format of the /EDIT command is:

```
/Edit  ┌filename┐  [,LRecl=number]
       │*       │
       └*2      ┘
```

filename
   specifies the name of an existing file which is to be changed. The file name parameter may be either a valid file name or a "*"

or "*2" indicating the appropriate temporary file. If the /EDIT command is entered with no file name the editor assumes a new file is being created.

LRECL

specifies the record length of the file being changed or created. If LRECL is not specified, and a new file is being created, the record length will default to 80. If LRECL is not specified, and an old file is being changed, the record length will be taken from the old file. If an old file is specified and an LRECL is coded which is smaller than the record length, the file will be truncated. The smallest LRECL value which can be specified is 20, the largest is 255.

<u>Notes:</u>

The largest file that can be edited is about 5000 records if the record length is 80 bytes (for files with a record length different from 80 bytes the record capacity will vary accordingly: fewer records for files with larger record lengths and vice versa). The reader is referred to the description of the /XEDIT command (page 15) for editing large library files.

General Description of Editor Operation

The editor is an interactive program which allows the user to create, inspect, and modify files. When the editor is used to modify an already existing file, it makes a temporary copy of that file, and all changes are made to that copy, using appropriate editor commands. The copy then replaces the original file when the user requests it (RSAVE, RFILE commands). Therefore, the user may terminate the edit session at any time without altering the original file (e.g., if erroneous modifications are made which would be too troublesome to reverse, or if second thoughts arise). Similarly, when the editor is used to create a new file, the file being created is not actually placed in the library until the user specifically requests it (SAVE, FILE commands), and the edit session may be terminated at any time without any new file actually being created. Termination of an edit session without the saving or replacing of a file may be achieved with either the QUIT command or with the system /CANCEL command (see the VPS Handbook).

The most commonly used editor commands allow the user to:

Type out part or all of the edited file (PRINT).

Search the file for lines containing specific character strings (FIND, LOCATE, SEARCH).

Add new lines to the file (INSERT, INPUT).

Delete or replace existing lines (DELETE, REPLACE).

Modify individual lines (CHANGE, BLANK, OVERLAY, ADD).

Move or copy groups of lines from one part of the file to another (MOVE, COPY).

This is by no means an exhaustive list of the editor´s capabilities and is intended merely to give the new user a feeling for what the editor is about.

The file being edited may be viewed as a sequence of records - the first record being called the "top" of the file, and a phantom record following the last record of the file the "bottom" of the file. This phantom record is used to terminate the action of several commands which are able to access or modify more than just one line of the file (e.g. CHANGE, PRINT, DELETE). When this line is reached, the message "*EOF" is printed, and the command terminates. In addition to the top and bottom of the file, some line in the file is always considered the "current line". This may be any line in the file, or it may be the phantom *EOF line - the current line may be printed out by simply entering the PRINT command with no parameters specified. Commands which act only on a single line of the file (e.g., one-line CHANGE commands, ADD, DELETE without parameters, REPLACE) act on the current line. The user may move around in the file (make another line the current line) by moving a specified number of lines toward the bottom of the file (the NEXT command) or toward the top of the file (the UP command) or by going directly to the top or bottom (TOP, BOTTOM commands). This motion may also be produced by requesting that the editor move in the desired direction to the nearest line containing some given character string or strings (FIND, LOCATE, UPFIND, UPLOCATE). Certain commands, such as multi-line CHANGE or DELETE commands, also cause a different line to become the current line.

The editor has two modes of operation - edit mode and input mode. In edit mode, lines typed in by the user are interpreted as editor commands (e.g. PRINT, CHANGE, FIND), and the user may perform the operations outlined above, such as moving around in the file, modifying lines of the file, and so forth. In input mode, on the other hand, all lines entered are simply put in the file. This is useful when more than one or two lines are to be added to the file, as when a new file is being created or a block of records is being placed in an existing file. Input mode is entered from edit mode via the INPUT command. Return to edit mode is accomplished by entering a null line - a line containing no characters. This null line may be sent by simply hitting carriage return in response to a keyboard read or, if a logical line delimiter character is defined (see the VPS Handbook), by entering the delimiter character as the first character of a physical input line or by placing two adjacent line delimiters in a physical input line. A blank line may be placed in the file in input mode by typing at least one blank before hitting carriage return. When input mode is entered, the editor responds with the message "INPUT", and when edit mode is entered, it responds with "EDIT". The "EDIT" message will also be typed out whenever the editor is in edit mode and a null line is entered. When /EDIT is first issued, the editor is placed in edit mode, unless the filename parameter has been omitted from the

/EDIT command. This is true whether or not the named file actually exists - if it does not exist, an informatory message is produced before edit mode is entered. If the filename parameter is omitted, the editor will be placed initially in input mode.

The user should also bear in mind that the record length of the edited file is fixed at the time the editor is invoked (/EDIT) and may not be changed during the edit. Therefore, if a file is to be given records longer than its current record length, the new record length must be specified on the /EDIT command (the LRECL parameter, see page 3). The record length of any library file may be altered by simply editing the file with the new record length specified and replacing the old copy with the new.

Syntax of Editor Commands

All editor commands begin with a keyword, such as PRINT, CHANGE, or LOCATE, which must appear as the first characters of the entered line. Most of the commonly used keywords may be abbreviated to minimize typing - P for PRINT, C for CHANGE, DEL for DELETE, and so forth. Some commands, such as TOP and BOTTOM, consist only of the keyword. Other commands, such as NEXT, DELETE, and PRINT, may be optionally given parameters which modify their actions, and still others, such as CHANGE, MOVE, and SPLIT, <u>must</u> be given parameters. One blank should appear between the command keyword and the parameter. The editor does not enforce this rule for all commands (e.g. "PRINT6" is equivalent to "PRINT 6"), but it does enforce it for commands which could otherwise allow ambiguities to arise (e.g. the abbreviation for INSERT is I, therefore, if the separating blank rule were not enforced for INSERT, the fairly common typo INPTU for INPUT would cause the line "NPTU" to be inserted into the file). The form of parameter which is appropriate for any given editor command depends on that command, but a few general remarks may be made about some of the elements which appear in these parameters.

Some commands, such as PRINT, NEXT, and DELETE, allow a parameter to be specified giving the number of lines to be acted upon (e.g. "PRINT 6" means print 6 lines starting with the current line). The only restriction imposed on this number is that it must not exceed 4 decimal digits, so that 9999 is the largest value that may be specified. This restriction also applies to the line number parameters which may be used with the MOVE and COPY commands.

A number of commands allow or require parameters which contain character strings. Depending on the command, a string must be in either the delimited or the non-delimited form. For commands requiring the non-delimited form, the character string is typed exactly as it is to be used by the editor. For example, the parameter expected by the INSERT command, which inserts a line into the file after the current line, consists simply of the contents of the line which is to be inserted. Thus, to insert the line "THE LINE" after the current line, we enter the command "INSERT THE LINE".

On the other hand, commands with a more complex syntax (e.g., requiring that more than one string be specified, allowing alternate non-string forms, or having both strings and other parameters) require that strings be typed in the delimited form. In this form, the user chooses a character to act as a string delimiter and types the string with one delimiter character at each end. If two or more contiguous strings are required by the command (e.g. the string form of MOVE), only a single delimiter character is typed between each pair of strings. The editor determines what character is being used as a string delimiter from the context of the command. The delimiter chosen must not appear in any of the strings to be delimited and, for DELETE, MOVE, and COPY, must not be numeric (i.e. 0-9). This latter restriction exists because the parameters for these commands have alternate numeric forms.

In the examples shown in this chapter, the slash, "/", is usually used as the string delimiter because it is in a handy place on the keyboard, and because it looks good. Obviously, if any slashes appear in the strings to be specified, some other character must be chosen for the delimiter. For example, the command to change the string "BUG" to "FIX" in the current line is "CHANGE /BUG/FIX/", where we have used the slash as the delimiter. If, on the other hand, we wish to change the string "X/2.0" to "X*.5" we could not use the slash. Thus, we might use "CHANGE $X/2.0$X*.5$".

On the Special Nature of TOP

It will occur to the reader, if it hasn't already, that if INSERT inserts a line into the file after the current line, and the top of the file is the first line of the file, then there seems to be no way to insert a line before the first line of the file. Well, fear not - if the INSERT command is issued immediately after a TOP command, the new line will be inserted before the first line of the file - if any command other than TOP appears between the TOP and the INSERT this special action will not occur. Thus the sequence

```
TOP
INSERT 123
```

inserts the line "123" before the first line of the file, and

```
TOP
PRINT
INSERT 123
```

inserts it after the first line.

The effect of an immediately preceding TOP is also felt by INPUT, GET, FIND, and LOCATE. In the case of INPUT and GET, the new lines will be added to the file before the first line of the file, and in the case of FIND and LOCATE the search will begin with the first, rather than the second, line of the file.

Verify and Brief Modes

The editor may be used in either verify mode or brief mode. In verify mode, the current line is printed or "verified" whenever it is altered (e.g., by CHANGE, ADD, OVERLAY) or whenever a new current line results from a request that the editor move around in the file (e.g., via UP, NEXT, FIND, UPLOCATE). In brief mode, this verification is suppressed. Unless brief mode has been specified in the user profile (see EDFLAG under /PROFILE in the VPS Handbook) the editor will always begin in verify mode. Brief mode is set by the BRIEF command, and verify mode by the VERIFY command. The VERIFY command may also be used to specify how many columns of the current line are to be typed out each time a verification occurs. Furthermore, a "flip character" may be defined and used to temporarily reverse the current verify/brief mode setting for any command which may cause verification (see below under "Command Suffixes").

Command Suffixes

A number of editor commands may be modified by the addition of a suffix to the command keyword. The three suffix types which exist are:

Cm.n where m and n designate the beginning and ending columns of the portion of the line or lines to be examined or modified by the command (see "The Zone" below for details). For example, the suffix C6.12 indicates that only columns 6 through 12 (inclusive) are to be treated. The default values for m and n, respectively, are 1 and the last character in the record (LRECL). Thus, if the edited file has records 80 bytes long, C10 means columns 10 through 80, and C.25 means columns 1 through 25.

L indicates that the command is in the so-called logical form, as described below under "Logical Expressions". If the Cm.n and L suffixes are allowed for a particular command, they are mutually exclusive.

flip character - a character which may be appended to the command keyword to reverse the current verify/brief setting for that instance of the command. This character may be set for any given edit session with the FLIP command, and a default flip character may be defined in the user profile (see FLIP under /PROFILE in the VPS Handbook). When a flip character is used in conjunction with a Cm.n or L suffix, it must follow that suffix.

As an example, consider the LOCATE command, which allows all three suffix types. This command may be abbreviated L, so

```
L XYZ
```

will cause the editor to look at each line of the file from the line following the current line to the bottom of the file until a line containing "XYZ" is found and then make that line the current line. If the editor is in verify mode, the new current line will be typed out. On the other hand, the modified command

```
          LC12.60 XYZ
```

will perform the same search except that only columns 12 through 60 of each line will be examined. If the user has set the flip character to "$", the typeout of the new current line caused by the above two commands will be suppressed if

```
          L$ XYZ
```

or

```
          LC12.60$ XYZ
```

respectively, is used instead.

The LOCATE command also has a logical form, invoked by using the L suffix. As described in detail below, a logical expression is an expression containing character strings and logical operators which has a value of either TRUE or FALSE for any given line of the file. The logical form of LOCATE is

```
          LL (logical expression)
```

This command is analogous to the string form of LOCATE discussed above, except that the search stops when a line is found for which the logical expression has a value of TRUE. As before, the typeout in verify mode may be suppressed by

```
          LL$ (logical expression)
```

The Zone

A number of commands, such as LOCATE, FIND, and the string form of DELETE, examine lines of the file for specified character strings, and other commands, such as CHANGE, ADD, and OVERLAY, modify lines of the file. As intimated above under "Command Suffixes", the action of these commands may treat the entire line or may be restricted to a portion of the line. If no Cm.n suffix is explicitly specified, the portion of the line to be acted upon is that specified by the current "zone" setting. When the editor is first invoked, the zone is set to the entire line from column 1 to the last column of each line (LRECL). The zone is changed with the ZONE command (q.v.). When a new zone is specified for a command by use of the Cm.n suffix as described above, the specification is in effect for that instance only.

Logical Expressions

A number of commands have alternate forms which employ logical expressions, allowing the user to specify more complex conditions on file searching and modification than would be possible using simple character strings. The basic element of a logical expression is the delimited string. A delimited string in a logical expression yields a value of TRUE or FALSE depending on whether it appears or does not appear in the line being examined (only the portion of the line defined by the current zone setting will be examined unless the zone is overridden as described below). These strings may be related by the binary logical operators AND, OR, and XOR (alternate forms &, |, and X), meaning and, inclusive or, and exclusive or, respectively. The unary operator NOT (alternate forms .NOT, ¬) may also be used. (Note that ASCII terminals on VPS use the circumflex, ^, for | and the tilde, ~, for ¬.) A logical expression must be enclosed in an outer set of parentheses, and additional sets of parentheses may be used within the expression to define sub-expressions.

The simplest logical expression contains only a single string. For example, the expression

```
(/FURD/)
```

will evaluate as TRUE for any line containing the string "FURD" within the currently defined zone. On the other hand, the expression

```
(/THUNDER/&/THUD/)
```

is TRUE only for lines containing <u>both</u> "THUNDER" and "THUD" within the current zone. Similarly,

```
(/THUNDER/|/THUD/)
```

is TRUE for lines containing <u>either</u> "THUNDER" or "THUD" within the current zone. Getting a bit more elaborate, the expression

```
(/JOKE/&(/FUNNY/|¬/PUN/))
```

is TRUE only if the zone contains "JOKE" and either contains "FUNNY" or does not contain "PUN".

The user may modify the effective zone for any string in a logical expression by appending a modifier term to it immediately after the second delimiter. The modifier term consists of either a Cm.n parameter or the parameter "F", or both, enclosed in parentheses. If both are given, they must be separated by a comma and may appear in either order. The Cm.n parameter specifies the portion of the line which is to be examined for that particular string, and F means that the string must start in the first column of the portion being examined. For example,

```
(/RED/(C10.30)&/BLUE/(F,C55))
```

is TRUE only for lines in which "RED" appears somewhere between columns 10 and 30 and "BLUE" appears starting in column 55. If F only is specified, it refers to the first column of the current zone. Thus, if the zone is set to columns 1 through 80, then

```
(/FROTZ/(F))
```

is TRUE only for lines which have "FROTZ" starting in column 1.

The string delimiter character is determined separately for each string in a logical expession. For example, if we need an expression which is TRUE for lines containing "THING" and "6.0/X" we cannot use the customary slash to delimit the second string. Thus, we might use

```
(/THING/&$6.0/X$)
```

Block Editing

In the general description of editor operation above we said that the editor creates a single temporary copy of the edited file, and that it is this temporary copy to which all modifications are applied during the edit session. While this simplified picture is adequate for the great majority of edits, it is occasionally useful to be able to juggle around two or more such temporary copies of the edited file or parts of the edited file, or other files or parts of files. Therefore, the editor is equipped with ten "blocks" called ":0", ":1", and so on, through ":9". (Note that some commands do not require a colon.) Each block is a separate editable entity and can contain any number of records, provided the number of records in all blocks combined does not exceed the capacity of the editor workspace. The record length of each block is the same as that of the originally edited file, as determined at the time the /EDIT command is issued, and cannot be altered during the session. At any given moment, some block is considered the "current block" - i.e., the block being edited, and all file searching and modification commands are applied to this block.

The SWITCH command (page 80) is used to make a different block the current block so that it may be edited. If the editor is in verify mode when SWITCH is issued, the current line in the switched-to block will be printed. Each block has its own current line, which is preserved if the user switches away from that block, and this line will become the active current line if the block is switched back to. The LIST command (page 57) may be used to type out a block other than the current block, and the GET command (page 51) may be used to copy a block other than the current block into the current block after the current line. Furthermore, the MOVE and COPY commands may be used to move or copy a group of records from the current block into another block after the current line in that block. The SIZE command (page 77) may be used to determine the number of lines in any block, and the = command (page 25) may be used to determine the position of the current line in any block.

When the editor is first invoked, the copy of the edited file is in block :0, and this block is the current block. All the other blocks, :1 through :9, are empty. Since it is used for the originally edited file, block :0 is called the principal block. If a SAVE, RSAVE, FILE, or RFILE command is issued, and the current block is not the principal block, :0, the message "WARNING - CURRENT BLOCK NOT PRINCIPAL, REPLY Y IF OK, N IF NOT" will be typed, and the user must indicate whether the command should proceed. The conversational read list is cleared before the user is polled, so any commands that had been entered after the SAVE, RSAVE, FILE, or RFILE command on the same physical input line are lost. While this feature is mildly annoying when one actually intends to put the contents of the current block in the library, the grief it saves on those rare occasions when absent mindedness takes one to the brink of disaster more than justifies it.

Editor Programs

It is sometimes necessary or desirable to make modifications to a file which require that a large number of the same or similar sequences of editor commands be executed. Since such repititious activities are best handled by machine, a facility has been put in the editor which allows the user to create and execute programs of editor commands. An editor program must reside in a block and consists of the sequence of commands which is to be executed. Each command must appear on a separate line in the same format as it would be typed if it were being issued directly by the user, except that the command keyword must be preceded by at least one blank or by a label beginning in column 1 and at least one separating blank.

Two editor commands exist which are meaningful only in editor programs: ENOP and EJMP. The ENOP command (page 44) is a dummy command used only to provide a place to hang a label when the user prefers not to label a normal command (such labels are a pain if another command has to be inserted between the label and the labeled command). The EJMP command (page 43) provides a conditional or unconditional branch to any label in the program.

An editor program is executed by means of the EXEC command (page 45), and one editor program may EXEC another, although recursion is not permitted. A program terminates when control passes beyond the last line of the program, or if an error occurs during its execution. Reaching the *EOF line is considered an error for this purpose (e.g., by issuing a NEXT or multiline CHANGE command).

As an example, the following program will delete every line containing an asterisk in column 1 (we assume the left boundary of the current zone is 1):

```
     TOP
LOOP ENOP
     EJMP (/*/(F)) DEL
     NEXT
     EJMP LOOP
DEL  ENOP
     DELETE
     EJMP LOOP
```

This program will terminate when *EOF is reached by either the NEXT or the DELETE command. Another program which would perform the same function is:

```
          TOP
     LOOP FIND *
          DELETE
          UP
          EJMP LOOP
```

except that this one might leave an asterisk line at the top of the file which had previously been the second line of the file (why?). This program will terminate when either the FIND fails to find a line beginning with an asterisk, or the DELETE causes *EOF to be reached. Naturally, the abbreviations for the various commands (F for FIND, DEL for DELETE, etc.) could have been used in these programs.

TRAN and NOTRAN

When lines of the file are typed out (e.g. with PRINT or as a result of verification) unprintable characters are normally replaced with blanks. The purpose of this translation is to prevent the untoward behavior of certain terminals, especially the more elaborate models with plotting modes and such, when the user is editing files containing characters outside the normal set of printable characters (e.g., object decks). This translation will be suppressed if NOTRAN (page 63) is issued and will be reinstated if TRAN (page 82) is issued.

Hexadecimal Editing

While each byte (character) of a file may take on any of 256 possible values, only about a third of these constitute printable characters. Therefore, to allow complete generality, the editor provides modes of operation in which data bytes are represented as two-digit hexadecimal numbers.

Two hexadecimal output modes are available: HEX and BOTH. When HEX is set (via the HEX command, page 53), lines typed out with PRINT or as a result of verification are displayed only in hex. For example, if the current line is "AB CD" and we issue PRINT with HEX in effect, the editor will type out

```
C1C240C3C4
```

When BOTH is set (BOTH, page 32), on the other hand, each line is displayed in both normal and hexadecimal form, with the character equivalent of each byte appearing above the left hand digit of the pair of digits representing that byte. If BOTH had been in effect when the above PRINT was issued, the editor would have responded with

```
A B   C D
C1C240C3C4
```

In BOTH mode, if a byte does not represent a printable character the corresponding position in the upper line is left blank. The ALPH command (page 29) is used to reset the output mode to the normal printable character form.

Input to the editor may also be specified in hexadecimal form. When XIN is issued, a preprocessor is activated which translates all lines typed in input mode and the parameters (operands) of all commands entered in edit mode from "generalized hexadecimal" form into a "result string" which is passed on to the editor for interpretation. The parameters of a command are assumed by the preprocessor to start with the first non-blank character after the blank which terminates the command keyword and any suffixes attached to it. In generalized hexadecimal form, bytes are represented by pairs of hexadecimal digits, and any number of blanks may appear between such pairs (the blanks are stripped off during translation). Furthermore, any non-blank character which is not a valid hexadecimal digit and which appears after an even number of hexadecimal digits is transferred to the result string unmodified, and any characters at all, including 0-9, A-F, and blanks, may be specified by enclosing them in apostrophes; apostrophes meant to appear in the result string being represented by double apostrophes in the usual way. Any number of such apostrophe-delimited strings may appear in an input line, and the line may slip into and out of apostrophe mode as many times as desired. Remember that blanks appearing outside apostrophes are deleted, while blanks appearing inside apostrophes are retained. If the remainder of a line is to be interpreted as within apostrophes, only the leading apostrophe is required - the closing apostrophe will be assumed.

For example, the following are all valid representations of the character string "/AB CD/"

```
/C1C2 40 C3 C4 /

'/AB CD/'

61C1C2 ' CD/

'/' C1C2 ' ' C3C4 '/'
```

Note that since the lines entered are processed before they are interpreted as commands a bit of caution is sometimes required. For example, a slash has the hexadecimal representation 61, so the command

```
CHANGE /61/40/
```

entered in XIN mode, would appear to the editor to be

```
CHANGE /// /
```

which is obviously invalid.

The input preprocessor is deactivated and normal input mode resumed by the AIN command (page 28).

/XEDIT - Editing Large Files

The /XEDIT command is used to run the same editor as the /EDIT command, except that the editor is given a larger workspace. This is useful when it is necessary to edit files larger than will fit in the normal editor workspace. The capacity of the editor when invoked by /XEDIT is about 27,000 records for a file with 80 byte records.

The format of the /XEDIT command is:

```
/XEdit  ┌filename┐  [,LRecl=number]
        │*       │
        └*2      ┘
```

where the parameters have the same meanings as those defined for /EDIT (page 3).

WEDT - Editing Work Files

A work file (see DISK files in the VPS Handbook) may be edited if its record length (LRECL) is not greater than 256 bytes. To edit such a file, one need only run the job:

```
/FILE UNIT=(4,DISK),...
/INC WEDT
```

where ... represents the rest of the parameters necessary to define the work file (DSNAME, VOL, DISP). The only difference between the operation of the editor when editing a work file and its normal operation is that FILE, RFILE, SAVE, and RSAVE commands issued without parameters cause the contents of the current block to be written to the work file rather than into the library. If a file name is given for any of these commands, the block contents will be placed in the library under that name, as usual.

Editor Commands

On the following pages a box -

```
 ________________________________________________________________
|                                                                |
|________________________________________________________________|
```

is used to enclose the definition of an editor command. The format of the information inside this box is defined as follows:

- The symbols [], {}, __, and ... are used to indicate how a command may be written. DO NOT CODE THESE SYMBOLS. Their general definitions are given below:

  [] indicates optional operands. The operand enclosed in brackets may or may not be coded, depending on whether or not the associated option is desired or whether a default is to be taken. If more than one item is enclosed in brackets, one or none of the items may be coded.

  {} indicates that a choice must be made. One of the operands from the vertical stack must be coded, depending on which of the associated functions is desired.

  __ indicates a value that is used in default of a specified value. This value is assumed if the operand is not coded.

  ... indicates that an operand, or set of operands, may be repeated. The number of repetitions is dependent upon the function desired.

- When a command has an abbreviated form the abbreviation will appear just below the command name. For example, in the ADD command description (page 26) the command name is written as -

  ADD
  A

  indicating that _either_ ADD or A may be used as the command name.

- Operands written entirely in lower case are used to denote that the user is to make the indicated substitution of a value, name, etc. For example, the definition of the NEXT command (page 62) is given as follows:

```
 ______________________________________________________
|                                                      |
|  NEXT  ┌number┐                                      |
|  N     │1     │                                      |
|        └      ┘                                      |
|______________________________________________________|
```

The user is instructed to replace 'number' with the actual number of lines to move (or take the default value of 1).

- Where the selected operand is a combination of upper and lower case letters - such as the column range operand of the CHANGE command (page 35) -

  ```
  ,Cm[.n]
  ```

  The user is to code the capital letter(s) exactly as shown and make the indicated substitutions for the lower case letters.

- Commas and delimiters should be coded exactly as shown, omitting only the comma following the last operand coded.

Note: Upper case letters are used in the command definitions only to differentiate between symbols and keywords required by the editor commands, and names or values selected by the user. If a command is being typed in on a terminal it should not be typed in upper case.

Column and Logical Forms

For commands having a column and/or logical form the definition of the special form can be found in the section describing the standard form of that command.

Example Formats

Two basic types of examples are used in the command descriptions. The first type consists of a description of a function followed by the command(s) necessary to perform that function, as in the EJMP command (page 43) and the SAVE command (page 73). The second type of example, and the most common, shows a line or set of lines in a file followed by a command followed by the lines as they were modified by that command. This is a ´before´ and ´after´ example. In these examples an arrow (-->) is used to show the current line before the execution of the command and the current line after the execution of the command. (Note, the editor does not print an arrow. The arrow is used in the examples to better clarify the operation of the command.) The examples used for the CHANGE command (page 35) and the INSERT command (page 55) are both of the second type. If a particular example is confusing, the user is encouraged to get on a terminal and try it.

Pg	Command	Abr	C	L	F	Function
21	$ASMCOM					Add comments to assembler source.
22	$ASMFIX					Column-align assembler source.
24	$SEQ					Place sequence numbers in cols. 73-80.
25	=					Print line number of current line.
26	ADD	A	X		X	Append characters to end of zone.
28	AIN					Set normal character input mode.
29	ALPH					Set normal character output mode.
30	BLANK	BL	X		X	Blank columns using mask.
32	BOTH					Set character and hex output mode.
33	BOTTOM	B				Go to bottom of file.
34	BRIEF	BR				Set brief mode.
35	CHANGE	C	X	X	X	Change character string in line(s).
38	COPY		X			Copy line(s) from one place to another.
40	DELETE	DEL	X	X	X	Delete line(s).
42	DUP					Duplicate current line.
43	EJMP					Branch in editor program.
44	ENOP					no-operation in editor program.
45	EXEC	E				execute editor program.
46	FILE					Write block to library or work file.
48	FIND	F	X		X	Find given string in 1st column of zone.
50	FLIP					set flip character.
51	GET					Read in block or library file.
52	GYRATE	G			X	Interchange current and next lines.
53	HEX					Set hex output mode.
54	INPUT		X			Enter input mode.
55	INSERT	I	X			Insert line into file.

Pg	Command	Abr	C	L	F	Function
56	JOIN	J			X	Join current and next lines.
57	LIST					List block or library file.
58	LOCATE	L	X	X	X	Locate line containing given string.
60	MOVE		X			Move line(s) from one place to another.
62	NEXT	N			X	Move down in file.
63	NOTRAN					Turn off output translator.
64	OVERLAY	O	X		X	Replace columns using mask.
65	PRINT	P				Type out lines of file.
66	QUIT	Q				Terminate edit without saving file.
67	REPEAT					Set repeat count for next BL or O.
68	REPLACE	R	X			Replace current line.
69	RFILE	RFIL				Replace lib. file with current block.
71	RSAVE	RSAV				RFILE and QUIT.
73	SAVE					FILE and QUIT.
75	SEARCH	S	X	X	X	Equivalent to TOP followed by LOCATE.
77	SIZE					Print number of lines in block.
78	SPLIT	SP	X		X	Divide current line into two lines.
80	SWITCH	SW			X	Switch to another block.
81	TOP	T				Go to top of file.
82	TRAN					Turn output translator on.
83	UNCHANGE	UNC			X	Undo modifications to current line.
84	UP	U			X	Move up in file.
85	UPFIND	UF	X		X	Like FIND but in upward direction.
87	UPLOCATE	UL	X	X	X	like LOCATE but in upward direction.
89	VERIFY	V				Set verify mode and length.
90	XIN					Set hex input mode.
91	ZONE	Z				Set default zone.

Columns:

Pg	- the page on which the command is decribed.
Command	- the command name.
Abr	- the command abbreviation (if any).
C	- X indicates this command has a column form.
L	- X indicates this command has a logical form.
F	- X indicates that the flip character may be used.

Suffixes:
    (none)

The $ASMCOM command may be used to add comments to assembler language source files. After the $ASMCOM command is issued in edit mode, the current line will be printed up to and including the last non-blank character on the line. The user may then type a comment (which normally must be preceded by a blank), and when finished, press carriage return. This sequence will then be repeated for each line remaining in the file. The user may type "$edit" instead of a comment to terminate $ASMCOM before the end of the file is reached. The standard form of the $ASMCOM command is shown below.

```
$ASMCOM
```

Suffixes:
(none)

The $ASMFIX command may be used to column justify an assembler language source file. The standard form of the $ASMFIX command is shown below.

```
| $ASMFIX  [inst column] , [oper column] , [com column]  |
|          [10         ]   [16         ]   [35        ]  |
```

inst column
is the column in which the assembler instruction field is to begin. The default is column 10.

oper column
is the column in which the operand field is to begin. The default is column 16.

com column
is the column in which the comment field is to begin. The default is column 35. If 73 is specified all comments will be deleted. If com column is greater than or equal to 74 all comments and comment lines will be deleted.

Note:
$ASMFIX will not column justify those lines in which one of the following characters apears in column 1:

/
*
.*
=

Also, $ASMFIX will not column justify object code. The command will, however, justify all other lines in the file including data lines, if they happen to be present.

Example:

1) In the following example a typical assembler language source file is column justified.

```
Before --> prog csect
            balr 12,0 establish addressability
            using *,12
            la 4,field begin process
            .
            .
            .

           $asmfix

After  --> prog     csect
                    balr  12,0                  establish addressability
                    using *,12
                    la    4,field               begin process
                    .
                    .
                    .
```

Suffixes:
(none)

The $SEQ command may be used to place sequence numbers in columns 73 through 80 of each line of a file. Normally, the first sequence number is 10 and the increment is 10. The standard form of the $SEQ is shown below.

$$\boxed{\texttt{\$SEQ}\quad\texttt{[identifier]}\ ,\begin{bmatrix}\texttt{n1}\\ \underline{\texttt{10}}\end{bmatrix}\ ,\begin{bmatrix}\texttt{n2}\\ \underline{\texttt{10}}\end{bmatrix}}$$

identifier
is a four character identifier which is to be placed in columns 73 through 76. If 'identifier' is omitted the sequence number will appear in the entire field padded to the left with zeroes.

n1
is the starting sequence number. The default is 10.

n2
is the increment. The default is 10.

Example:

1)
```
        la    r4,1(,r4)
        lr    r8,r7
        b     start
        .
        .
        .

  $seq ardv

        la    r4,1(,r4)                                  ardv0010
        lr    r8,r7                                      ardv0020
        b     start                                      ardv0030
        .
        .
        .
```

Suffixes:
(none)

The = command may be used to find the current line number. In other words, if the number 8 was returned when the = command was issued, the current line is the eighth line in the file. The standard form of the = command is shown below.

```
=    [block number]
```

block number
is the number of the editor block in which the number of the current line is to be listed. If block number is not specified the number of the current line in the current editor block is listed.

Suffixes:
Column
Flip character

The ADD command may be used to add a character string to the end of the current line. The standard form of the ADD command is shown below.

```
ADD   string
A
```

string
is a string of characters to be added to the end of the current line immediately following the last non-blank character. Note that the ADD command assumes that the first character in the string is one space past the command name, i.e. -

```
String Start         String Start
          ↓                     ↓
      ADD                    A
```

Column Form:

```
ADDCm[.n]   string
ACm[.n]
```

The column form of ADD may be used to add a string after the last non-blank character within the column range ´m´ and ´n´. Note that the string will be truncated if it falls past column ´n´.

Examples:

1) In the following example the string ´8910´ is added to the current line.

```
--> 1234567

    a 8910

--> 12345678910
```

2) In the following example the character ´s´ is added after column 4.

```
 --> bird  of paradise

     ac4.5 s

 --> birds of paradise
```

Suffixes:
(none)

Normal input to the editor is in character form, however, the XIN command (see page 90) may be used to allow hexadecimal input. The AIN command is used to reset the editor to character input mode. The standard form of the AIN command is shown below.

```
AIN
```

Suffixes:
(none)

Normal printed output from the editor is in the form of characters, or text. However, the HEX (see page 53) or BOTH (see page 32) commands may be used to produce hexadecimal output. The ALPH command is used to reset the editor to character output mode. The standard form of the ALPH command is shown below.

```
ALPH
```

Suffixes:
 Column
 Flip character

The BLANK command may be used to set one or more columns of the current line to blanks. The standard form of the BLANK command is shown below.

```
 BLANK   string
 BL
```

string
 is blank and non-blank characters. For each non-blank character of string, the corresponding column of the current line is set to blank. Note that column 1 of string is one space past the command name, i.e. -

```
   Column 1          Column 1
          ↓                 ↓
    BLANK              BL
```

Column Form:

```
BLANKCm[.n]   string
BLCm[.n]
```

The column form of BLANK allows the user to begin the blanking action at column ´m´ rather than column 1. ´n´ specifies the column in which the blanking action is to be terminated.

Examples:

1) In the following example the current line is blanked in columns 1, 2, 6, and 8.

```
 --> apples are usually red

     bl zz   z z

 -->   ple   re usually red
```

2) In the following example the current line is blanked in columns 10 and 13.

```
 --> how now brown cow

     blc10.13 z  z

 --> how now b ow  cow
```

Suffixes:
(none)

The BOTH command may be used to specify that both character and hexadecimal representations of lines are to be printed in output mode. To reset the editor to character output mode the reader is referred to the ALPH command (page 29). The standard form of the BOTH command is shown below.

```
BOTH
```

Suffixes:
(none)

The BOTTOM command may be used to move to just after the last line in the file (i.e., to *EOF). The standard form of the BOTTOM command is shown below.

```
BOTTOM
B
```

Suffixes:
(none)

The BRIEF command may be used to discontinue the editor action of listing referenced and changed lines for certain commands such as, CHANGE, NEXT, UP, etc. (see also the VERIFY command - page 89). The standard form of the BRIEF command is shown below.

```
BRIEF
BR
```

Suffixes:
- Flip character
- Logical

The CHANGE command may be used to replace one character string with another character string, of the same or a different length, on a line or set of lines. The editor will automatically expand or compress the line relative to the replacement string length. If the replacement character string causes the line to extend beyond the file line length, excess characters will be truncated. The standard form of the CHANGE command is shown below.

```
CHANGE   /string1/string2/┌count┐ [,G]  [,V]  [,F]  [,Cm[.n]]
C                         └*    ┘

                          [,P]
```

/ (slash)
: signifies any unique delimiting character that does not appear in the character strings involved in the change.

string1
: is the character string to be replaced. If ´string1´ is null (i.e., c //string2/...) then the replacement string will be placed preceding the first character of the line.

string2
: is the character string which is to replace ´string1´. If ´string2´ is null (i.e., c /string1//...) then ´string1´ will be deleted from the line.

count
: is the number of lines, starting with the current line, in which this change is to be made. ´*´ indicates that the change is to all lines from the current line to the bottom line of the file.

G
: specifies that each occurrence of ´string1´ in a line is to be replaced. Note that if G is not specified, only the first occurrence of ´string1´ will be replaced.

V
: specifies that the editor is to list all changes if multiple lines are being changed. Note that if V is not specified and ´count´ is

specified, only the last changed line will be listed.

F

specifies that the replacement is to be made only if ´stringl´ starts in the first column of the zone (normally, column 1).

Cm[.n]

specifies that the replacement is to occur only if ´stringl´ is between columns ´m´ and ´n´ of the line. If this parameter is specified, any expanding, compressing, or truncating will take place within the given column range.

P

specifies that the editor is to query the user before each replacement of a multiline change is made. The valid replies to the query are:

y - indicates that the change should be made.

n - indicates that the change should not be made.

s - indicates that the change should not be made and that the command should be terminated, returning to edit mode.

= - indicates that the line number should be printed.

v - indicates that verify mode should be set to on.

Logical Form:

```
CHANGEL (expr) /stringl/string2/┌count┐ [,G]  [,V]  [,F]  [,Cm[.n]]
CL                              └*    ┘

                                        [,P]
```

The logical form of the CHANGE command is the same as the standard form except the logical expression ´(expr)´ must test TRUE for the change to occur (for a description of logical expressions see page 10).

Examples:

```
1) --> four and seven years ago

       c /four/four score/

   --> four score and seven years ago
```

```
2) --> turnips are good and radishes are good

       c /good/good also/c35.80

   --> turnips are good and radishes are good also

3) --> do i=1 to n;
          array(i,j)=array(i,j)+array(i,2);
          end;

       c /i/k/3,g

       do k=1 to n;
          array(k,j)=array(k,j)+array(k,2);
   -->    end;
```

Suffixes:
  Column

The COPY command may be used to copy one or more lines of a file to another location within the file. The standard form of the COPY command is shown below.

```
COPY { {n3,            }  n1,n2                  }
     { {:block number, }                          }
     {                                            }
     { {/string3       }  /string1/string2/[F]    }
     { {:block number, }                          }
```

n3,n1,n2
:block number,n2,n3

where ´n3´ is the line number after which the copied lines are to be inserted, ´n1´ is the first line number of the lines to be copied, and ´n2´ is the last line number of the lines to be copied.

If a colon (:) followed by a block number is specified instead of ´n3´, the lines will be copied into the indicated editor block and inserted after the current line in that editor block. Note, line numbers may be found using the = command (see page 25).

/string3/string1/string2/
:block number,/string1/string2/

where / (slash) is any unique delimiting character (<u>other than</u> 0 through 9 and :) which does not appear in the character strings. ´string3´ is a character string. The lines to be copied will be placed after the line containing the first occurrence of ´string3´ when searching from the top of the file. ´string1´ is a character string. The first line to be copied will be the line containing the first occurrence of ´string1´ when searching from the top of the file. ´string2´ is also a character string. The last line to be copied will be the line containing the first occurrence of ´string2´ when searching from the top of the file.

If a colon (:) followed by a block number is specified instead of ´string3´, the lines will be copied into the indicated editor block and inserted after the current line in that editor block. Note that the line containing ´string2´ must be found at least one line below the line containing ´string1´. F is optional and indicates that the string searches must only begin in the first column of the current zone (normally, column 1).

Column Form:

```
COPYCm.[n]    /string3/string1/string2/[F]
```

The column form of the COPY command may be used to search for the three strings only within the column range ´m´ through ´n´. F implies that the search will begin only in column ´m´.

Example:

1) Before:

```
    .
    .
    .
    x = 0;                         <-- assuming this is line number 408
    y = 0;
    call action(x,y);
    put skip edit (x,y) (2 f(5.2));
    .
    .
    .

   copy 411,408,409
```

After:

```
    .
    .
    .
    x = 0;
    y = 0;
    call action(x,y);
    put skip edit (x,y) (2 f(5.2));
    x = 0;
    y = 0;
    .
    .
    .
```

Suffixes:
- Column
- Flip character
- Logical

The DELETE command may be used to delete one or more lines from a file. The standard form of the DELETE command is shown below.

```
DELETE  ┌number  ┐
DEL     │1       │
        └/string/┘
```

number
: is the number of lines starting with the current line which are to be deleted. If no parameter is specified, 1 will be used as the default - deleting the current line.

/string/
: where / (slash) is any unique delimiting character not appearing in ´string´ nor the digits 0 through 9. ´string´ is a character string and indicates that all lines from the current line down to, but not including, the next line containing the character string will be deleted.

Column Form:

```
DELETECm[.n]   /string/
DELCm[.n]
```

The column form of the DELETE command may be used to search for the character string only in the column range ´m´ through ´n´.

Logical Form:

```
DELETEL   (expr)
DELL
```

The logical form of the DELETE command may be used to delete all lines from the current line down to, but not including, the first line for which the logical expression ´(expr)´ is TRUE (for a description of logical expressions see page 10).

Note:
In all forms of the DELETE command, after the command has completed the current line will be the line after the last line deleted from the file.

Examples:

```
1) --> the brown cow ate grass, jumped
       over the fence, and took
       a bus to Detroit.

       del 2

   --> a bus to Detroit.

2) -->  x = 0;
        y = 0;
        call action(x,y);

       delc14 /x/

   -->  call action(x,y);
```

Suffixes:
Column

The DUP command may be used to duplicate the current line one or more times. The standard form of the DUP command is shown below.

```
DUP  ┌number┐
     │1     │
     └      ┘
```

number
is the number of times the current line is to be duplicated. 1 is the default.

Column Form:

```
DUPCm[.n]  number
           1
```

The column form of the DUP command may be used to duplicate the information in the current line which appears within the column range ´m´ through ´n´.

Note:
After the DUP command has completed, the current line will be the last duplicated line.

Examples:

```
1) -->  a(i) = a(i) + 2;  i = i + 1;

       dup 2

        a(i) = a(i) + 2;  i = i + 1;
        a(i) = a(i) + 2;  i = i + 1;
   -->  a(i) = a(i) + 2;  i = i + 1;

2) -->  x = x + 2;  x = x + 1;

       dupc12.40 2

        x = x + 2;  x = x + 1;
                    x = x + 1;
   -->              x = x + 1;
```

Suffixes:
(none)

The EJMP command is used to cause a conditional or unconditional branch to be taken within an editor program (for a description of editor programs see page 12). The standard form of the EJMP command is shown below.

```
[label]   EJMP   [(expr)]   {jump label}
```

(expr)
is a logical expression which when true will cause the branch to be taken (for a description of logical expressions see page 10).

jump label
is the label of the command to which the branch is to be made.

Note:
If no logical expression is supplied, the branch is unconditional and will always be made.

Example:

1) The following editor program would print every fifth line of a file:

```
loop n 5
     ejmp loop
```

Suffixes:
(none)

The ENOP command may be used as a ´no operation´ facility for editor programs (similar to CONTINUE in FORTRAN and ANOP in Assembler Macro language). For a description of editor programs, the reader is referred to page 12. The standard form of the ENOP command is shown below.

```
[label]   ENOP
```

Example:

1) The following editor program would print every fifth line of a file:

```
loop enop
     n 5
     ejmp loop
```

Suffixes:
(none)

The EXEC command may be used to execute an editor program (for a description of editor programs see page 12). The standard form of the EXEC command is shown below.

```
EXEC   {block number}
E
```

block number
is the number of the block containing the editor program to be executed.

Suffixes:
(none)

The FILE command is used to save the contents of the current editor block (normally, the edited file) as a new VPS library file without terminating the edit session (to replace an existing library file see the RFILE command, page 69). The standard form of the FILE command is shown below.

```
FILE   filename [,accattr,accattr,...]
```

filename

is any 1 to 8 character name under which the edited file (or contents of the current editor block) is to be saved. If the file is to be created under an account different from the user´s account, the file name must be immediately followed (no intervening blanks) by the owning account number enclosed in parentheses (this requires a privilege).

accattr

specifies an access attribute the user wishes to change (note that if this parameter is not specified default access attributes will be taken from the user´s profile). It may be specified as a single attribute or a list of attributes separated by commas. The access attributes permitted are:

READ (R) - specifies that the file may be read or copied by other accounts.

NOREAD (NOR) - specifies that the file may not be read or copied by other accounts.

EXEC (E) - specifies that the file may be the argument of the /EXEC command (the file may be executed) by other accounts.

NOEXEC (NOE) - specifies that the file may not be the argument of the /EXEC command (the file may not be executed) by other accounts.

PURGE (P) - specifies that the file may be updated, edited, replaced, or purged by other accounts.

NOPURGE (NOP) - specifies that the file may not be updated, edited, replaced, or purged by other accounts.

P=(A=acctrange,accattr) - specifies the account or account range to be given special access to the file and the access attributes allowed for these account(s). ´acctrange´ may be specified as a single account; a range of accounts separated by a dash ("-"), in which case all accounts within that range will be given the special access attributes; a "partial account" (represented by a partial account number followed by an asterisk ("*")), indicating that all accounts beginning with the "partial account" should be given the special access attributes; or, finally, a range of "partial accounts" separated by a dash, indicating that all accounts starting with the first account beginning with the first "partial account" to the last account beginning with the second "partial account" should be given the special access attributes. ´accattr´ is any of the access attributes listed above.

Alternatively, the following access attributes may also be specified:

PUBL - which corresponds to "READ", "EXEC", and "NOPURGE" access.
PRIV - which corresponds to "NOREAD", "NOEXEC", and "NOPURGE" access.
XO - which corresponds to "NOREAD", "EXEC", and "NOPURGE" access.
NOXO - which corresponds to "NOEXEC" access.

Examples:

1) In the following example a current copy of the edited file is saved under the new VPS library file name - BUMPERS:

```
.
.
.
file bumpers
*SAVED
.
.
.
```

2) In the following example a current copy of the edited file is saved under the new VPS library file name - JUMPERS - and the access attribute NOEXEC is specified:

```
.
.
.
file jumpers,noe
*SAVED
.
.
.
```

Suffixes:
Column
Flip character

The FIND command may be used to move to the next line in the file which contains the specified character string starting in the first column of the current zone (normally, column 1). The standard form of the FIND command is shown below.

```
FIND   {string}
F
```

string
is a character string. Note that the FIND command assumes that the first location in the string is one space past the command name, i.e. -

```
String Start            String Start
          ↓                        ↓
     FIND                    F
```

Column Form:

```
FINDCm[.n]   {string}
FCm[.n]
```

The column form of the FIND command may be used to search for the character string starting in column ´m´ rather than the first column of the current zone.

Note:
If the FIND command is unsuccessful in searching for the character string, the current line remains the same.

Examples:

```
1) -->            lr     r7,r8
                  la     r6,nextone
                  b      start
       nextone    ds     0h
                  .
                  .
                  .

       f next

                  lr     r7,r8
                  la     r6,nextone
                  b      start
   --> nextone    ds     0h
                  .
                  .
                  .

2) -->            lr     r7,r8
                  la     r6,nextone
                  b      start
       nextone    ds     0h
                  .
                  .
                  .

       fc19 next

                  lr     r7,r8
   -->            la     r6,nextone
                  b      start
       nextone    ds     0h
                  .
                  .
                  .
```

Suffixes:
(none)

The FLIP command may be used to define an active flip character which may be appended to a subset of the editor commands (in this manual any command allowing the flip character is indicated as such under the subheading Suffixes). If the active flip character is appended to one of these commands, the current verify/brief mode setting will be reversed for that command execution. The standard form of the FLIP command is shown below.

```
 FLIP   {symbol}
```

symbol
  is any character which is to be used as the currently active flip character.

Examples:
Assuming that the current verify/brief mode setting is verify, note the difference between the following command sequences:

Command Sequence 1:

```
top
f the cow
THE COW JUMPED OVER THE MOON
```

Command Sequence 2:

```
top
flip $
f$ the cow
p
THE COW JUMPED OVER THE MOON
```

Suffixes:
  (none)

The GET command is used to copy part or all of a VPS library file or an editor block into the file being edited. The standard form of the GET command is shown below.

```
GET   [n1,n2,] {filename      }
               {:block number }
```

n1,n2
: where ´n1´ is the line number of the first line in the file (or block) to be copied and ´n2´ is the line number of the last line in the file (or block) to be copied. If ´n1´ and ´n2´ are omitted, the entire file or block is copied. Note, the copied lines will be placed after the current line.

filename
: is the name of the VPS library file to be copied.

block number
: is the number of the editor block to be copied. Note that the block number must be preceded by a colon (:).

Example:

1) Suppose editor block number 2 contains the following:

```
q=y;
do i = 1 to 10;
  x(i) = 0;
  end;
```

In this example lines 2 through 4 of the editor block are copied into the file.

```
-->  b = q;

    get 2,4,:2

     b = q;
     do i = 1 to 10;
      x(i) = 0;
-->   end;
```

Suffixes:
 (none)

The GYRATE command may be used to physically move the current line, placing it below the next line in the file. The current line remains the current line. The standard form of the GYRATE command is shown below.

```
GYRATE
G
```

Example:

```
1) -->  x = a + b;
        a = a + 1;

       g

        a = a + 1;
   -->  x = a + b;
```

Suffixes:
(none)

The HEX command is used to specify that only the hexadecimal representations of lines are to be printed in output mode. To reset the editor to character mode the reader is referred to the ALPH command (page 29). The standard form of the HEX command is shown below.

```
HEX
```

Suffixes:
    Column

The INPUT command is used to enter input mode so that new lines may be added to a file. The new lines will be added after the current line (note that if the command immediately preceding INPUT was TOP, the lines will be added before the first line of the file). To return to edit mode from input mode the user should enter a ´null´ line (containing only a carriage return). The standard form of the INPUT command is shown below.

```
|-------------------------------------------------------------|
|  INPUT                                                      |
|_____________________________________________________________|
```

Column Form:

```
   INPUTCm[.n]
```

The column form of the INPUT command may be used to enter new lines into a file but placed in the column range ´m´ through ´n´ rather than starting in the first column of the current zone (normally, column 1).

Examples:

```
1) -->  do i = 1 to 10;

       input
       INPUT
          x(i) = 0;
          end;
                                    <-- Null line (carriage return)

        do i = 1 to 10;
          x(i) = 0;
   -->    end;

2) -->  do i = 1 to 10;

       inputc5
       INPUT
       x(i) = 0;
       end;
                                    <-- Null line (carriage return)

        do i = 1 to 10;
           x(i) = 0;
   -->     end;
```

Suffixes:
  Column

The INSERT command may be used to enter a new line in the file after the current line (note that if the command immediately preceding INSERT was TOP, the line will be inserted before the first line of the file). The standard form of the INSERT command is shown below.

```
INSERT   {new line}
I
```

new line
    is any string of characters to be added as a separate line in the file. Note that the INSERT command assumes that the first location in the new line is one space past the command name, i.e. -

```
  String Start          String Start
             ↓                     ↓
      INSERT                      I
```

Column Form:

```
INSERTCm[.n]   {new line}
ICm[.n]
```

The column form of the insert command may be used to insert a new line into a file but placed in the column range ´m´ through ´n´ rather than starting in the first column of the current zone.

Examples:

```
1) -->  z = x + y;

       i  r = a * b;

        z = x + y;
   -->  r = a * b;

2) -->  z = x + y;

       ic4 r = a * b;

        z = x + y;
   -->     r = a * b;
```

Suffixes:
    Flip character

The JOIN command may be used to append the next line to the current line after the last non-blank character. The standard form of the JOIN command is shown below.

```
 JOIN   [/string/]
 J
```

/ (slash)
    is any unique delimiting character that does not appear in the character string.

string
    is a character string which is to be inserted between the end of the current line and the information appended from the next line. If ´string´ is not specified the first location in the next line will be placed immediately following the last non-blank character of the current line.

Examples:

```
1) --> abc
       12345

       j

   --> abc12345

2) --> dear mom,
       send money

       j / howdy, /

   --> dear mom, howdy, send money
```

Suffixes:
(none)

The LIST command may be used to print all or part of a VPS library file or another editor block at the user's terminal while editing a file. The standard form of the LIST command is shown below.

```
LIST  [num1,       ] [num2,       ] {filename       }
      [1,          ] [LAST,       ] {:block number  }
      [LAST,       ] [LAST-num3,  ]
      [LAST-num3,  ]
```

first parameter
specifies the line number in the file where printing is to begin. The first line in the file being line number 1. The parameter may be specified either as a number (represented by "num1" above) relative to the beginning of the file; as the keyword LAST, indicating that the last line in the file is to be printed; or as the keyword LAST followed by a minus sign ("-") and a number (represented by "num3" above), indicating that the first line to be printed is line number "LAST-num" relative to the end of the file. Line number 1 is the default.

second parameter
specifies the line number in the file where printing is to end. The parameter may be specified as a number (represented by "num2" above) relative to the beginning of the file; as the keyword LAST, indicating the printing should continue until the end of the file is reached; or as the keyword LAST followed by a minus sign ("-") and a number (represented by "num3" above), indicating that the last line to be printed is line number "LAST-num" relative to the end of the file. If only the first parameter is specified, it is used as the default for the second parameter (i.e., only one line will be printed). If neither parameter is specified, LAST is used as the default for the second parameter.

filename
is the name of the VPS library file to be listed.

block number
is the number of the editor block to be listed. Note that the block number must be preceded by a colon (:).

<u>Suffixes</u>:
- Column
- Flip character
- Logical

The LOCATE command may be used to move to the next line in the file which contains the specified character string anywhere in the current zone (normally, the entire line). The standard form of the LOCATE command is shown below.

```
LOCATE   {string}
L
```

string
    is a character string. Note that the LOCATE command assumes that the first location in the string is one space past the command name, i.e. -

```
  String Start          String Start

     LOCATE                    L
```

<u>Column Form</u>:

```
LOCATECm[.n]   {string}
LCm[.n]
```

The column form of the LOCATE command may be used to search for the character string in the column range ´m´ through ´n´ rather than the current zone.

<u>Logical Form</u>:

$$\begin{matrix}\text{LOCATEL} \\ \text{LL}\end{matrix} \begin{bmatrix}*, \\ \text{number,}\end{bmatrix} \text{(expr)}$$

The logical form of the LOCATE command may be used to locate a line under the control of a logical expression ´(expr)´ (for a description of logical expressions see page 10). The search is successful when the next line is found for which the logical expression is TRUE.

If ´*´ is specified <u>all</u> lines (from the next line to the bottom of the file) for which the logical expression is TRUE will be <u>listed</u> regardless of the verify/brief mode setting. In this special case of the LOCATE command the current line <u>remains unchanged</u>.

If a number (´number´) is specified preceding the logical expression, only those lines for which the logical expression is TRUE from the next line for ´number´ lines will be listed regardless of the verify/brief mode setting. In this special case of the LOCATE command the current line <u>remains</u> <u>unchanged</u>.

<u>Note</u>:
In all forms of the LOCATE command (except the two special cases noted above) if the search for the character string is successful or if the logical expression is satisfied, the line containing the character string or satisfying the logical expression becomes the current line. If the search is unsuccessful the current line remains the same.

<u>Examples</u>:

```
1) -->           lr    r7,r8
                 la    r6,nextone
                 la    r6,1(,r6)
                 b     start

       l start

                 lr    r7,r8
                 la    r6,nextone
                 la    r6,1(,r6)
   -->           b     start

2) -->           lr    r7,r8
                 la    r6,nextone
                 la    r6,1(,r6)
                 b     start

       lc22.23 r6

                 lr    r7,r8
                 la    r6,nextone
   -->           la    r6,1(,r6)
                 b     start
```

Suffixes:
Column

The MOVE command may be used to move one or more lines of a file to another location within the file or to another editor block. The standard form of the MOVE command is shown below.

```
MOVE ⎧⎧n3,            ⎫  n1,n2                ⎫
     ⎪⎩:block number, ⎭                       ⎪
     ⎨                                        ⎬
     ⎪⎧/string3       ⎫  /string1/string2/[F] ⎪
     ⎩⎩:block number, ⎭                       ⎭
```

n3,n1,n2
:block number,n1,n2

where ´n3´ is the line number after which the moved lines are to be inserted, ´n1´ is the first line number of the lines to be moved, and ´n2´ is the last line number of the lines to be moved.

If a colon (:) followed by a block number is specified instead of ´n3´, the lines will be moved into the indicated editor block and inserted after the current line in that editor block. Note, line numbers may be found using the = command (see page 25).

/string3/string1/string2/
:block number,/string1/string2/

where / (slash) is any unique delimiting character (other than 0 through 9 and :) which does not appear in the character strings. ´string3´ is a character string. The lines to be moved will be placed after the line containing the first occurrence of ´string3´ when searching from the top of the file. ´string1´ is a character string. The first line to be moved will be the line containing the first occurrence of ´string1´ when searching from the top of the file. ´string2´ is also a character string. The last line to be moved will be the line containing the first occurrence of ´string2´ when searching from the top of the file.

If a colon (:) followed by a block number is specified instead of ´string3´, the lines will be moved into the indicated editor block and inserted after the current line in that editor block. Note that the line containing ´string2´ must be found at least one line below the line containing ´string1´. F is optional and indicates that the string searches must only begin in the first column of the current zone (normally, column 1).

Column Form:

```
MOVECm.[n]   /string3/string1/string2/[F]
```

The column form of the MOVE command may be used to search for the three strings only within the column range ´m´ through ´n´. F implies that the search will begin only in column ´m´.

Example:

1) Before:

```
     .
     .
     .
     x = 0;                       <-- assuming this is line number 408
     y = 0;
     call action(x,y);
     put skip edit (x,y) (2 f(5.2));
     .
     .
     .

    move 411,408,409
```

After:

```
     .
     .
     .
     call action(x,y);
     put skip edit (x,y) (2 f(5.2));
     x = 0;
     y = 0;
     .
     .
     .
```

Suffixes:
  Flip character

The NEXT command may be used to have the editor move down one or more lines. The standard form of the NEXT command is shown below.

```
 NEXT  [number]
 N     [1     ]
```

number
  is the number of lines the editor is to move. The default is 1.

Example:

```
1) -->  do i = 1 to 10;
           x(i) = 0;
           end;

       n 2

        do i = 1 to 10;
           x(i) = 0;
   -->     end;
```

Suffixes:
(none)

The NOTRAN command is used to set no-translation mode for output printing. In no-translation mode special characters appearing in files will be printed on the terminal (normally, these special characters are translated out before printing). To reset translation mode see the TRAN command, page 82. The standard form of the NOTRAN command is shown below.

```
NOTRAN
```

Suffixes:
  Column

The OVERLAY command may be used to overlay a line or portion of a line with the characters of a string. The standard form of the OVERLAY command is shown below.

```
OVERLAY   {string}
O
```

string
  is a character string. Each non-blank character of ´string´ will replace the corresponding character of the current line. Note that the OVERLAY command assumes that the first location in the string, corresponding to the first column of the current zone (normally, column 1), is one space past the command name, i.e. -

```
  String Start        String Start
             ↓                   ↓
     OVERLAY                    O
```

Column Form:

```
OVERLAYCm[.n]   {string}
OCm[.n]
```

The column form of the OVERLAY command is similar to the standard form except that one space past the command name corresponds to column ´m´ rather than the first column of the current zone.

Examples:

```
1) --> 1234567890

       o a b c

   --> a2b4c67890

2) --> 1234567890

       oc4.10 a b c

   --> 123a5b7c90
```

Suffixes:
(none)

The PRINT command may be used to list one or more lines starting with the current line. The standard form of the PRINT command is shown below.

```
PRINT  ┌number┐  [,length]
P      └1     ┘
```

number
is the number of lines to be listed. The default is 1. Note that after the PRINT command has completed, the current line will be the last line listed.

length
is a number representing the number of columns starting with column 1 which are to be listed in each line. If ´length´ is omitted all columns of the lines are listed.

Examples:

```
1) --> dear mom,
       send money

       p 2
       DEAR MOM,
       SEND MONEY

2) --> dear mom,
       send money

       p 2,4
       DEAR
       SEND
```

Suffixes:
    (none)

The QUIT command may be used to terminate an editor session without saving the edited file. The terminal session is returned to '*go'. The standard form of the QUIT command is shown below.

```
QUIT
Q
```

Suffixes:
(none)

The REPEAT command is used to repeat a following BLANK or OVERLAY command one or more times, moving down the file one line after each BLANK or OVERLAY command execution. The standard form of the REPEAT command is shown below.

```
REPEAT  [number]
        [1     ]
```

number
is the number of times the BLANK or OVERLAY command is to be executed. The default is 1.

Note:
After a REPEAT command has been issued it will remain in effect until a BLANK or OVERLAY command is issued or another REPEAT command is issued, irrespective of what comes in between.

Example:

```
1) -->          la     4,1(,4)                                   00000400
                lr     6,7                                       00000410
                b      next                                      00000420

      repeat 3
      blc73.80 xxxxxxxx

                la     4,1(,4)
                lr     6,7
   -->          b      next
```

Suffixes:
 Column

The REPLACE command is used to replace the current line with the contents of a character string. The standard form of the REPLACE command is shown below.

```
REPLACE    {string}
R
```

string
 is a character string which is to replace the current line. Note that the first location in the string, corresponding to the first column in the current zone (normally, column 1), is one space past the command name, i.e. -

```
  String Start          String Start
             ↓                     ↓
     REPLACE                     R
```

Column Form:

```
REPLACECm[.n]    {string}
RCm[.n]
```

The column form of the REPLACE command may be used to replace only the information on the current line between the column range ´m´ through ´n´.

Examples:

```
1) -->  x = a * b;

        r  y = z + x;

   -->  y = z + x;

2) -->  x = a * b;

        rc6.10 z + y

   -->  x = z + y;
```

Suffixes:
(none)

The RFILE command is used to save the contents of the current block (normally, the edited file) in an existing VPS library file without terminating the edit session (to create a new library file see the FILE command, page 46). The standard form of the RFILE command is shown below.

```
RFILE   [filename] [,accattr,accattr,...]
RFIL
```

filename
is any 1 to 8 character name which the current edited file is to replace. If the file exists under an account different from the user´s account, the file name must be immediately followed (no intervening blanks) by the owning account number enclosed in parentheses (this requires a privilege). If the file name is not specified the editor will attempt to replace the file whose name was on the /EDIT command.

accattr
specifies an access attribute the user wishes to change (note that if this parameter is not specified the existing access attributes of the file will remain unchanged). It not specified default access attributes will be taken from the user´s profile). It may be specified as a single attribute or a list of attributes separated by commas. The access attributes permitted are:

READ (R) - specifies that the file may be read or copied by other accounts.
NOREAD (NOR) - specifies that the file may not be read or copied by other accounts.
EXEC (E) - specifies that the file may be the argument of the /EXEC command (the file may be executed) by other accounts.
NOEXEC (NOE) - specifies that the file may not be the argument of the /EXEC command (the file may not be executed) by other accounts.
PURGE (P) - specifies that the file may be updated, edited, replaced, or purged by other accounts.
NOPURGE (NOP) - specifies that the file may not be updated, edited, replaced, or purged by other accounts.
P=(A=acctrange,accattr) - specifies the account or account range to be given special access to the file and the access attributes allowed for these account(s). ´acctrange´ may be specified as a single account; a range of accounts separated by a dash ("-"), in which case all accounts within that range will be given the special access attributes; a

"partial account" (represented by a partial account number followed by an asterisk ("*")), indicating that all accounts beginning with the "partial account" should be given the special access attributes; or, finally, a range of "partial accounts" separated by a dash, indicating that all accounts starting with the first account beginning with the first "partial account" to the last account beginning with the second "partial account" should be given the special access attributes. ´accattr´ is any of the access attributes listed above.

Alternatively, the following access attributes may also be specified:

PUBL - which corresponds to "READ", "EXEC", and "NOPURGE" access.
PRIV - which corresponds to "NOREAD", "NOEXEC", and "NOPURGE" access.
XO - which corresponds to "NOREAD", "EXEC", and "NOPURGE" access.
NOXO - which corresponds to "NOEXEC" access.

Examples:

1) In the following example a current copy of the edited file replaces the VPS library file - BUMPERS:

```
/edit bumpers
.
.
.
rfil
*REPLACED
.
.
.
```

2) In the following example a current copy of the edited file replaces the VPS library file - JUMPERS - and the access attribute NOEXEC is specified:

```
.
.
.
rfil jumpers,noe
*REPLACED
.
.
.
```

<u>Suffixes</u>:
(none)

The RSAVE command is used to save the contents of the current block (normally, the edited file) in an existing VPS library file <u>and</u> terminate the edit session (to create a new file see the SAVE command, page 73). The standard form of the RSAVE command is shown below.

```
RSAVE    [filename] [,accattr,accattr,...]
RSAV
```

filename
is any 1 to 8 character name which the current edited file is to replace. If the file exists under an account different from the user´s account, the file name must be immediately followed (no intervening blanks) by the owning account number enclosed in parentheses (this requires a privilege). If the file name is not specified the editor will attempt to replace the file whose name was on the /EDIT command.

accattr
specifies an access attribute the user wishes to change (note that if this parameter is not specified the existing access attributes of the file will remain unchanged). It not specified default access attributes will be taken from the user´s profile). It may be specified as a single attribute or a list of attributes separated by commas. The access attributes permitted are:

READ (R) - specifies that the file may be read or copied by other accounts.
NOREAD (NOR) - specifies that the file may not be read or copied by other accounts.
EXEC (E) - specifies that the file may be the argument of the /EXEC command (the file may be executed) by other accounts.
NOEXEC (NOE) - specifies that the file may not be the argument of the /EXEC command (the file may not be executed) by other accounts.
PURGE (P) - specifies that the file may be updated, edited, replaced, or purged by other accounts.
NOPURGE (NOP) - specifies that the file may not be updated, edited, replaced, or purged by other accounts.
P=(A=acctrange,accattr) - specifies the account or account range to be given special access to the file and the access attributes allowed for these account(s). ´acctrange´ may be specified as a single account; a range of accounts separated by a dash ("-"), in which case all accounts within that range will be given the special access attributes; a "partial account" (represented by a partial account number

followed by an asterisk ("*")), indicating that all accounts beginning with the "partial account" should be given the special access attributes; or, finally, a range of "partial accounts" separated by a dash, indicating that all accounts starting with the first account beginning with the first "partial account" to the last account beginning with the second "partial account" should be given the special access attributes. ´accattr´ is any of the access attributes listed above.

Alternatively, the following access attributes may also be specified:

PUBL - which corresponds to "READ", "EXEC", and "NOPURGE" access.
PRIV - which corresponds to "NOREAD", "NOEXEC", and "NOPURGE" access.
XO - which corresponds to "NOREAD", "EXEC", and "NOPURGE" access.
NOXO - which corresponds to "NOEXEC" access.

Examples:

1) In the following example a current copy of the edited file replaces the VPS library file - BUMPERS:

```
/edit bumpers
.
.
.
rsav
*REPLACED
*GO
.
.
.
```

2) In the following example a current copy of the edited file replaces the VPS library file - JUMPERS - and the access attribute NOEXEC is specified:

```
.
.
.
rsav jumpers,noe
*REPLACED
*GO
.
.
.
```

<u>Suffixes</u>:
(none)

The SAVE command is used to save the contents of the current block (normally, the edited file) as a new VPS library file <u>and</u> terminate the edit session (to replace an existing library file see the RSAVE command, page 71). The standard form of the SAVE command is shown below.

```
SAVE   filename [,accattr,accattr,...]
```

filename
is any 1 to 8 character name under which the edited file (or contents of the current editor block) is to be saved. If the file is to be created under an account different from the user´s account, the file name must be immediately followed (no intervening blanks) by the owning account number enclosed in parentheses (this requires a privilege).

accattr
specifies an access attribute the user wishes to change (note that if this parameter is not specified default access attributes will be taken from the user´s profile). It may be specified as a single attribute or a list of attributes separated by commas. The access attributes permitted are:

READ (R) - specifies that the file may be read or copied by other accounts.
NOREAD (NOR) - specifies that the file may not be read or copied by other accounts.
EXEC (E) - specifies that the file may be the argument of the /EXEC command (the file may be executed) by other accounts.
NOEXEC (NOE) - specifies that the file may not be the argument of the /EXEC command (the file may not be executed) by other accounts.
PURGE (P) - specifies that the file may be updated, edited, replaced, or purged by other accounts.
NOPURGE (NOP) - specifies that the file may not be updated, edited, replaced, or purged by other accounts.
P=(A=acctrange,accattr) - specifies the account or account range to be given special access to the file and the access attributes allowed for these account(s). ´acctrange´ may be specified as a single account; a range of accounts separated by a dash ("-"), in which case all accounts within that range will be given the special access attributes; a "partial account" (represented by a partial account number followed by an asterisk ("*")), indicating that all accounts beginning with the "partial account" should be given the

special access attributes; or, finally, a range of "partial accounts" separated by a dash, indicating that all accounts starting with the first account beginning with the first "partial account" to the last account beginning with the second "partial account" should be given the special access attributes. ´accattr´ is any of the access attributes listed above.

Alternatively, the following access attributes may also be specified:

PUBL - which corresponds to "READ", "EXEC", and "NOPURGE" access.
PRIV - which corresponds to "NOREAD", "NOEXEC", and "NOPURGE" access.
XO - which corresponds to "NOREAD", "EXEC", and "NOPURGE" access.
NOXO - which corresponds to "NOEXEC" access.

Examples:

1) In the following example a current copy of the edited file is saved under the new VPS library file name - BUMPERS:

```
.
.
.
save bumpers
*SAVED
*GO
.
.
.
```

2) In the following example a current copy of the edited file is saved under the new VPS library file name - JUMPERS - and the access attribute NOEXEC is specified:

```
.
.
.
save jumpers,noe
*SAVED
*GO
.
.
.
```

Suffixes:
- Column
- Flip character
- Logical

The SEARCH command may be used to move to the first line in the file which contains the specified character string regardless of the location of the current line (equivalent to issuing a TOP command, page 81, followed by a LOCATE command, page 58). The standard form of the SEARCH command is shown below.

```
SEARCH   {string}
S
```

string
: is a character string. Note that the SEARCH command assumes that the first location in the string is one space past the command name, i.e. -

```
String Start          String Start
           ↓                     ↓
    SEARCH                     S
```

Column Form:

```
SEARCHCm[.n]   {string}
SCm[.n]
```

The column form of the SEARCH command may be used to search for the character string in the range ´m´ through ´n´ rather than the current zone.

Logical Form:

```
SEARCHL  ┌*,      ┐  (expr)
SL       └number, ┘
```

The logical form of the SEARCH command may be used to search for a line under the control of a logical expression ´(expr)´ (for a description of logical expressions see page 10). The search is successful only when the logical expression is TRUE.

If ´*´ is specified all lines for which the logical expression is TRUE will be listed regardless of the verify/brief mode setting. In this special case of the SEARCH command the current line remains

<u>unchanged</u>.

If a number (´number´) is specified preceding the logical expression, only those lines for which the logical expression is TRUE within the range of line numbers 1 to ´number´ will be listed regardless of the verify/brief mode setting. In this special case of the SEARCH command the current line <u>remains</u> <u>unchanged</u>.

<u>Note</u>:
In all forms of the SEARCH command (except the two special cases noted above) if the search for the character string is successful or if the logical expression is satisfied, the line containing the character string or satisfying the logical expression becomes the current line. If the search is unsuccessful the current line remains the same.

<u>Examples</u>:

```
1)                lr    r7,r8
                  la    r6,nextone
                  la    r6,1(,r6)
   -->            b     start

        s r6

                  lr    r7,r8
   -->            la    r6,nextone
                  la    r6,1(,r6)
                  b     start

2)                lr    r7,r8
                  la    r6,nextone
                  la    r6,1(,r6)
   -->            b     start

        sc22 r6

                  lr    r7,r8
                  la    r6,nextone
   -->            la    r6,1(,r6)
                  b     start
```

Suffixes:
  (none)

The SIZE command is used to determine the number of lines in a file (or block) being edited. The standard form of the SIZE command is shown below.

```
SIZE
```

Suffixes:
Column

The SPLIT command may be used to split a line into two separate lines at a specified character string. The line created from the right hand piece is inserted after the remainder of the original line and becomes the new current line. The standard form of the SPLIT command is shown below.

```
 ____________________________________________________________
|                                                            |
|  SPLIT   /string/    Cm[.n]   [,ICq]                       |
|  SP                                                        |
|____________________________________________________________|
```

/ (slash)
signifies any unique delimiting character that does not appear in the character string.

string
is a character string. All characters up to, but not including, the first occurrence of the character string will remain unchanged. The remainder of the line, including the character string, will be moved to a new line which becomes the current line.

Cm[.n]
specifies that the search for ´string´ is to be between columns ´m´ and ´n´ of the line.

ICq
where ´q´ is the column number in which the split-off chunk is to begin. If this parameter is not specified the first character of ´string´ will be placed in the first column of the current zone (normally, column 1).

Examples:

```
1)       do i = 1 to 10;
  -->      x(i) = 0;     end;

       sp /     e/

         do i = 1 to 10;
           x(i) = 0;
  -->      end;
```

```
2) --> next      la    r2,1(,r2)

       sp /la/ic10

       next
   -->           la    r2,1(,r2)
```

Suffixes:
    (none)

The SWITCH command is used to switch from one editor block to another. The standard form of the SWITCH command is shown below.

```
 ___________________________________________________________
|                                                           |
|  SWITCH   {block number}                                  |
|  SW                                                       |
|___________________________________________________________|
```

block number
    is the number of the block to which the user wishes to switch.

Notes:
If the new block is empty, the editor will respond with ´*eof´. If the new block contains lines of information and the editor is in verify mode, the editor will list the current line in the block. Further information on block editing can be found on page 11.

Suffixes:
(none)

The TOP command may be used to have the editor move to the top of the file. The standard form of the TOP command is shown below.

```
TOP
T
```

Suffixes:
(none)

The TRAN command is used to reset the editor to translation mode. In translation mode special characters appearing in lines of a file are translated to blanks before they are printed (the characters in the file are not changed). This is the normal editor setting. The standard form of the TRAN command is shown below.

```
TRAN
```

Suffixes:
(none)

The UNCHANGE command is used to nullify the effect of changes made to the current line with such commands as ADD, CHANGE, OVERLAY, etc. The UNCHANGE command is effective only if the editor has not moved away from the current line since the unwanted changes were made. Note that internal motion will also cause the UNCHANGE command to be ineffective (e.g., an unsuccessful LOCATE). The standard form of the UNCHANGE command is shown below.

```
UNCHANGE
UNC
```

Example:

```
1) --> life is a drag

       c /drag/bowl of cherries/
       LIFE IS A BOWL OF CHERRIES
       unc

   --> life is a drag
```

Suffixes:
    Flip character

The UP command may be used to have the editor move up one or more lines in a file. The standard form of the UP command is shown below.

```
 UP   [number]
 U    [1     ]
```

number
    is the number of lines the editor is to move. The default is 1.

Example:

```
1)              la     r4,l(,r4)
                lr     r7,r8
   -->          b      start

       u 2

   -->          la     r4,l(,r4)
                lr     r7,r8
                b      start
```

Suffixes:
Column

The UPFIND command may be used to move to the first line above the current line which contains the specified character string starting in the first column of the current zone (normally, column 1). The standard form of the UPFIND command is shown below.

```
UPFIND   {string}
UF
```

string
is a character string. Note that the UPFIND command assumes that the first character in the string is one space past the command name, i.e. -

```
  String Start         String Start
             ↓                    ↓
      UPFIND                  UF
```

Column Form:

```
UPFINDCm[.n]   {string}
UFCm[.n]
```

The column form of the UPFIND command may be used to search for the character string starting in column ´m´ rather than the first column of the current zone.

Note:
If the UPFIND command is unsuccessful in searching for the character string, the current line remains unchanged.

Examples:

```
1)      nextone  ds     0h
                 lr     r7,r8
                 la     r6,nextone
   -->           b      start
                 .
                 .
                 .

        uf next

   --> nextone   ds     0h
                 lr     r7,r8
                 la     r6,nextone
                 b      start
                 .
                 .
                 .

2)               lr     r7,r8
                 la     r6,nextone
                 b      start
   --> nextone   ds     0h
                 .
                 .
                 .

        ufcll r

   -->           lr     r7,r8
                 la     r6,nextone
                 b      start
        nextone  ds     0h
                 .
                 .
                 .
```

Suffixes:
  Column

The UPLOCATE command may be used to move to the first line above the current line which contains the specified character string anywhere in the current zone (normally, the entire line). The standard form of the UPLOCATE command is shown below.

```
UPLOCATE   {string}
UL
```

string
  is a character string. Note that the UPLOCATE command assumes that the first location in the string is one space past the command name, i.e. -

```
  String Start          String Start
             ↓                     ↓
   UPLOCATE                    UL
```

Column Form:

```
UPLOCATECm[.n]   {string}
ULCm[.n]
```

The column form of the UPLOCATE command may be used to search for the character string in the column range ´m´ through ´n´ rather than the current zone.

Note:
  In all forms of the UPLOCATE command if the search for the character string is successful, the line containing the character string becomes the current line. If the search is unsuccessful the current line remains unchanged.

Examples:

```
1)                lr    r7,r8
                  la    r6,nextone
                  la    r6,1(,r6)
   -->            b     start

        ul next

                  lr    r7,r8
   -->            la    r6,nextone
                  la    r6,1(,r6)
                  b     start

2)                lr    r7,r8
                  la    r6,nextone
                  la    r6,1(,r6)
   -->            b     start

        ulcll r

   -->            lr    r7,r8
                  la    r6,nextone
                  la    r6,1(,r6)
                  b     start
```

Suffixes:
(none)

The VERIFY command is used to set the editor to verify mode so that referenced and changed lines will be listed (see also the BRIEF command, page 34). Note that verify mode is the default editor mode setting (unless the user's profile indicates otherwise). The standard form of the VERIFY command is shown below.

```
VERIFY   [number]
V
```

number
is the number of columns to be printed when listing referenced and changed lines. If number is not specifed, all columns of the current zone will be printed.

<u>Suffixes</u>:
(none)

The XIN command is used to set the editor to hexadecimal input mode. In hexadecimal input mode all input strings are assumed to be in generalized hexadecimal form (see page 13). To reset the editor to character input mode see the AIN command, page 28. The standard form of the XIN command is shown below.

```
XIN
```

Suffixes:
(none)

The ZONE command is used to change the editor zone. Initially, the zone is from column 1 to the logical record length (the entire line) indicating that all commands will operate on all columns of the line (except, of course, for the column form of appropriate commands). The ZONE command can be used to change this default zone (or reset it); in which case subsequent editor commands will operate only on the specified zone - as if nothing else existed on the lines. The standard form of the ZONE command is shown below.

```
ZONE   [m.n]
Z
```

m.n
where ´m´ is the first column number of the new zone and ´n´ is the last column number of the new zone. If ´m.n´ is not specified, the editor will reset the zone to the full line.

Examples:

```
1) -->          ltr   r4,r4
                bnz   start
                la    r4,1(,r4)
       start    ds    0h

       z 1.8
       l start

                ltr   r4,r4
                bnz   start
                la    r4,1(,r4)
   --> start    ds    0h

2) -->  x(i) = x(i+1) + 1;

       z 8.20
       c /i/a/

   -->  x(i) = x(a+1) + 1;
```

Chapter 2: /SCRIPT - the Text Processor

Script* is a program that allows the user to print English language text in a form suitable for term papers, theses, publications, resumes, etc. The Script program formats a file containing text and Script control lines and produces an edited listing. The file must first be typed in on a terminal, using the VPS Editor, and saved in the VPS library. Control lines entered in the file along with the text allow the user to control page numbering, top and bottom margin specifications, left and right margin specifications, underlining, etc.

The /SCRIPT Command

The /SCRIPT command invokes the Script program to format the text file. This command may be used either at a terminal or in a VPS batch job. If run in the VPS batch, the formatted listing will be printed on the batch printer. The format of the /SCRIPT command for both terminal and batch is shown below.

```
/SCRIPT    {filename}    [,option,option,....]
```

filename
: specifies the name of the library file the user wishes to have formatted and printed.

option
: specifies a /SCRIPT control option. It may be specified as a single option or as a list of options separated by commas. The following is a list of the control options available:

  CEnter - specifies that the listing should be centered on 132 position printout paper.
  NOwait - specifies that the printing of the text should begin immediately without pausing for the user to adjust the paper.
  NUmber - specifies that the name of the text file and the line number of each line should be listed next to the left margin.

----

* The VPS Script program is based on the M.I.T. NSCRIPT text processor. Parts of this chapter have been taken from the NSCRIPT Reference Guide, Copyright 1973, Massachusetts Institute of Technology, and are used with the permission of M.I.T. Information Processing Services.

OFfline - specifies that though this script execution is being run at the terminal, printer control characters are to be used in the scripted output. This option should be used if a /FILE statement has been defined for unit 6 setting the unit type to PRINTER and the file is being scripted at the terminal.

PAGExxx - specifies that scripting should begin at page "xxx". Note "xxx" must be entered as three numeric digits (e.g., PAGE003).

SIngle - specifies that the listing should terminate after one page is printed.

STop - specifies that the listing of the text should stop after each page is printed. This allows the insertion of standard typewriter paper if desired.

UNformatted - specifies that the file should be listed without formatting.

TRanslate - specifies that character translation should occur using a translation table that may be present as part of the text file.

2PASS - specifies that the /SCRIPT program should make one pass through the text file performing all indicated formatting without producing any output. A second pass is then made, during which output is produced. This option is primarily useful for handling a text file which uses reference names defined later in the file.

Using /SCRIPT From Terminals

After the /SCRIPT command, with the appropriate options, has been entered at the user´s terminal, the following message will appear:

```
vps/nscript
load paper; hit return.
```

Script will now wait until the user positions the paper in the terminal. The paper should be positioned one line above the top of the page to be printed. Script will begin printing the formatted listing when the user presses the carriage return key. If single sheets of typewriter paper are used instead of continuous form terminal paper, the user should specify the STOP parameter on the /SCRIPT command (see above) so that Script will allow the user to load new paper between pages of formatted text.

Using /SCRIPT From the Batch

To print a scripted listing on the batch printer, the following batch job format should be used:

```
/ID  account
/SCRIPT  filename,option,option,...
/END
```

If the user wishes to place the scripted listing into a special output class for, say, special forms, then the following batch job format should be used (note that in this example the scripted listing would be placed in sysout class S):

```
/ID account
/FILE UNIT=(6,PRINTER),CLASS=S,OPT=NOSKIP
/SCRIPT filename,option,option,...
/END
```

Notes on the Behavior of /SCRIPT under VPS

/INCLUDE control statements (see the VPS Handbook) may not be used in script files. Script supports two control words which allow the appending of files (.AP, see page 100) and the imbedding of files (.IM, see page 128) during /SCRIPT processing. These control words should be used to perform the functions of /INCLUDE control statements.

When underlining (see the .UL control word, page 162) or overprinting (see the .BS control word, page 105) /SCRIPT sends three characters to the terminal for each character location on the print line (e.g., when underlining the characters sent are - the character to be underlined, a control character to physically backspace the terminal, and the underscore character). Therefore, when doing heavy underlining or overprinting it may be necessary to increase the linesize value known to VPS; since VPS will fold lines longer than the current linesize - causing interesting (though not necessarily what the user intended) script listings. The linesize may be increased by using the /CTL command (see the VPS Handbook). The following would set the linesize to 250 characters:

```
/CTL LINESIZE 250
```

Script Text Formatting

The following is a list of the Script program defaults which were chosen to print text with margin control on standard 8 1/2 by 11 inch typewriter paper using an IBM 2741 10-pitch font:

Top Margin:	5 blank lines
Bottom Margin:	3 blank lines
Left Margin:	Defined by position of paper
Right Margin:	60 spaces to the right of the left margin, e.g., line length of 60 spaces
Lines per page:	66 lines

These default settings may be changed by using Script control words described later in this chapter. Note that this manual was produced with Script on an AJ 832 using a 12-pitch font and the following settings:

Top Margin:	7 blank lines
Bottom Margin:	4 blank lines
Left Margin:	Defined by position of paper

```
Right Margin:    70 spaces to the right of the left margin, e.g.,
                 line length of 70 spaces
Lines per page: 66 lines
```

Once Script has determined the margin settings it then manipulates the text to fit within these margins. Two methods are used to adjust the text - concatenation and justification. Concatenation is the process of joining together consecutive sentences, or sentence fragments, so that no short or long lines will appear in the body of a paragraph. For example, if the original text is entered as:

```
This is an example
of the concatenation
feature
of Script.
```

Script will concatenate these lines to read:

```
This is an example of the concatenation feature of Script.
```

Justification is the insertion of extra blank spaces between words in a line of text so that the last letter of the last word will always be printed at the right margin column, insuring a straight right margin. For example, if the original text was entered as:

```
This is an example of the Script justification feature
```

The formatted Script listing would look like:

```
This   is    an   example   of   the   Script  justification   feature
```

Certain special conditions exist for the Script concatenation and justification features. The last line of a paragraph will not be justified as it is a valid short line. Any line of text which begins with a blank space will cause concatenation and justification to terminate with the previous line and to begin anew with the line containing the blank. This feature, called a "break", prevents Script from concatenating several individual paragraphs into one long paragraph. An explicit break may also be caused by the break command word (.br). Implicit breaks are caused by any of the control words that cause normal line spacing to be interrupted (control words which cause a break are designated as such in the control word descriptions later in this chapter).

Preparing Text for Script

The successful use of Script is based on certain conventions that must be followed when preparing the text. The first line of a paragraph must be indented at least one space from the left margin unless blank lines or break control words (e.g., .BR) are inserted between paragraphs. Subsequent lines in a paragraph must be single spaced in the user's text. Double spaced text listings can be obtained by use of a special Script control word.

Due to the concatenation feature of Script, the use of hyphens (-) to break words between syllables is not recommended. If a word is too long to fit on a line it should be typed on the next line without hyphenation. Hyphens used to join two words, e.g., left-justify, may be used with Script.

Script Control Lines

Script control lines may be inserted into the text that is to be processed by Script. These control lines are not the same as the options specified on the /SCRIPT command. The format of a Script control line follows.

```
.cw   parameter(s)
```

Each control word (".cw" above) must start with a period ("."), identifying this line as a control line, and must begin in column 1. The control word is followed by user specified parameters, if needed. Control lines are not output, but are interpreted to specify the format of the output. Control lines must be entered as single lines in the file and the control word may be specified in either upper or lower case.

Control lines may appear anywhere in the file, and have effect on all output produced after their appearance. Input data should not be placed on a control line, as it may be erroneously interpreted as a parameter (except in the case of the .UR control word).

Script Control Words

The following pages describe in detail the control words which may be used with Script. The symbols [], {}, __, and ... are used to indicate how a control line may be written. DO NOT CODE THESE SYMBOLS. Their general definitions are given below:

[] indicates optional operands. The operand enclosed in brackets may or may not be coded, depending on whether or not the associated option is desired or whether a default is to be taken. If more than one item is enclosed in brackets, one or none of the items may be coded.

{} indicates that a choice must be made. One of the operands from the vertical stack must be coded, depending on which of the associated services is desired.

__ indicates a value that is used in default of a specified value. This value is assumed if the operand is not coded.

.. indicates that an operand, or set of operands, may be repeated. The number of repetitions is dependent upon the function desired.

*: Causes a break

CW	PURPOSE	PAGE
.AP	Append	100
.AR	Arabic page numbering	101
.BL	Blank	102
.BM	* Bottom Margin	103
.BR	* Break	104
.BS	Backspace	105
.CE	* Center next line	106
.CM	Comment	107
.CN	Create index line	108
.CO	* Concatenate	110
.CP	* Conditional page	111
.DC	Don´t count	112
.DS	* Double space	113
.EM	Empty page	114
.FD	Odd-page footing	115
.FE	Even and odd page footing	116
.FM	* Footing margin	117
.FN	Footnote definition	118
.FO	* Format	120
.FV	Even page footing	121
.HD	Odd page heading	122
.HE	Even and odd page heading	123
.HM	* Heading margin	125
.HV	Even page heading	126
.IF	Conditional	127
.IM	Imbed	128
.IN	* Indent	130
.JU	* Justify	131

CW	PURPOSE	PAGE
.LL	* Line length	132
.LS	Leading blank line	133
.NC	* No concatenate	134
.NF	* No format	135
.NJ	* No justify	136
.OF	* Offset	137
.PA	* Page eject	139
.PD	Odd page force	140
.PL	* Page length	141
.PN	Page numbering	142
.PR	Print terminal message	143
.PV	Even page force	144
.RA	* Right adjust	145
.RC	Read control	146
.RD	Read	147
.RM	Remote	148
.RO	Roman numeral page numbering	150
.SP	* Space	151
.SR	Set reference name	152
.SS	* Single space	155
.TB	Set output tab locations	156
.TC	Set tab character	157
.TM	* Top margin	158
.TR	Specify translate table	159
.TS	Triple space	161
.UL	Underline	162
.UN	* Undent	163
.UR	Use reference name	164

The APPEND control word allows an additional file to be appended to the file just printed.

```
.AP  ┌indexname.filename┐   ┌    *    ┐
     └USERLIB.filename  ┘   │n1  ─  n2│
      ───────               └n1     * ┘
```

When the .AP control word is encountered, the current file will be closed. Line numbers "n1" through "n2" of the "filename" will be read and printed as a continuation of output already produced. If an index name is not specified, USERLIB is assumed.

Defaults:

The entire file is read in starting at the first line and no break is created.

Notes:

(1) "*" may be coded in place of "n1 n2" to indicate that the entire file is to be read. If no arguments appear after the filename, "*" is assumed.
(2) If "n1" is coded, "*" may be coded in place of "n2" to indicate that the entire file starting with item "n1" is to be read. If the third argument is omitted, "*" is assumed.
(3) The .AP control word only allows files to be appended to the end of the current file. If it is desired to insert file contents at some point besides the end, the .IM control word should be used.
(4) /INCLUDE control statements (see the VPS Handbook) may not be used with script files. .AP or .IM (see page 128) should be used instead.

Examples:

(a) .ap cha4cont
The file named CHA4CONT will be read and formatted for output as a continuation of the current file.

(b) .ap fig2 12 *
The file named FIG2 will be read starting with item 12 and output as a continuation of the current file.

The ARABIC control word causes page numbers to use arabic numerals.

```
.AR
```

The .AR control word causes all page numbers produced hereafter in headings or footings to be printed in arabic numerals.

Defaults:

Page numbering in arabic is in effect until an .RO is encountered. A break is not created by this command.

Notes:

(1) Arabic numerals are the normal mode.

The BLANK control word allows a character to be specified as the blank replacement character.

```
| .BL          [character]                                      |
```

Each blank control character in the file will be replaced with a blank. Since justification is done before substitution the resulting blank space(s) will not be expanded by justification. This control character is especially useful when a specific number of blanks is required between two words or symbols and Script is right and left justifying the text.

Defaults:

Unless otherwise specified, no blank character is defined.

Example:

(1) The sequence:

```
.bl $
(1)$This will cause only 1 blank to appear in
")$T" though this line
will be justified by Script.
```

Produces:

```
(1) This will  cause only 1 blank  to appear in  ") T" though
this line will be justified by Script.
```

The BOTTOM MARGIN control word specifies the number of lines which are to appear between the bottom of the output page and the last line of ordinary or footnote text.

```
.BM          [n]
             [1]
```

At the bottom of all subsequent output pages (including the current page), _n_ lines will appear between the bottom of printed text (footnote text included) and the physical bottom of the page.

Any footing line will appear within these _n_ lines.

Defaults:

This command does create a break. Unless otherwise specified _n_ = 3 will be in effect. When this command word is encountered and the operand is omitted _n_ = 1 will be assumed.

Notes:

(1) See also the description of the .FM (Footing Margin) control word (page 117).

(2) At no time may the value set in .BM be smaller or equal to the value set in .FM.

BREAK causes the immediately previous line to be typed without filling in with words from the next line.

```
 .BR
```

BREAK is used to prevent concatenation of lines such as paragraph headings or the last line of a paragraph. It causes the preceding line to be typed as a short line if it is shorter than the current line length.

Notes:

(1) Many of the other control words act as a BREAK. No BREAK is necessary when one of these is present. In the description of each control word it states, in the section titled Defaults, whether that control word acts as a break or not.

(2) A leading blank or tab on a line has the effect of a BREAK immediately before the line.

(3) If NO CONCATENATE is in effect, all lines appear to be followed by a BREAK.

(4) If a BREAK control word immediately follows a .IF control word (see page 127) which tests FALSE the BREAK is ignored.

Example:

(1)

```
 Heading:
 .br
 First line of paragraph ...
```

This part of the file will be printed as:

```
 Heading:
 First line of paragraph ...
```

If the BREAK control word were not included, it would be typed:

```
 Heading: First line of paragraph ...
```

The BACK SPACE control word allows a character to be specified as the back space or overstrike character.

```
.BS          [character]
```

Back space characters present in the file will cause Script to back space one character position before printing the next character found.

A .BS control word with no operands will reset the back space character to the default.

Defaults:

Unless otherwise specified the back space character is the EBCDIC backspace character (hexadecimal 16).

Notes:

(1) The back space character is never printed in the formatted output listing.
(2) See also the description of the .UL (Underline) control word (page 162).
(3) It may be necessary to increase the terminal linesize value when doing heavy overprinting with the .BS character. The VPS /CTL command (see the VPS Handbook) should be used. The following would set the linesize value to 250 characters:

```
/ctl linesize 250
```

Example:

(1) The sequence:

```
.bs @
This is an example of b@_a@_c@_k@__s@_p@_a@_c@_i@_n@_g@_
```

Produces:

This is an example of back spacing

The line following the CENTER control word will be centered between the margins.

```
 .CE
```

The next line in the input file, including its leading blanks, will be centered between the left and right margins.

Defaults:

This command does cause a break. Any input lines or parts of lines that have not been typed will be typed without filling in words from any following lines.

Notes:

(1) If the line to be centered is longer than the current line length, it will be truncated and not centered.

(2) The left and right margins are the value of any indent value (.IN) and the current line length, respectively.

Example:

(1)
```
.ce
Other Methods
```

When this line is typed, the characters "Other Methods" will be centered:

```
                    Other Methods
```

The COMMENT control word is ignored and may be used to enter comments into a script file.

```
.CM         [anything]
```

The .CM control word allows comments to be stored in the SCRIPT file. These comments may be seen whenever the file is printed (using /LIST, /DISPLAY, /EDIT or the UNFORMATTED option).

Comment lines may be used to store unique identifications for use during editing of the file.

Defaults:

This command does not cause a break.

Notes:

(1) Since only the first two characters of a control line (exclusive of the ".") are examined to determine what control word is present, and since undefined reference names have null-string ("") values, a reference name can be given the value ´CM´ and used to control conditional recognition of other control words.

Examples:

(1) .cm .nf used below.

The comment line will be seen when examining the unformatted printout of the script file and will serve to explain what is going on to the reader.

The CREATE INDEX control word specifies three items of index information to be written to the *2 file (see the VPS Handbook).

```
.CN        ´s1´s2´s3´
```

Where "s1", "s2" and "s3" are character strings not containing quotation marks. Any of the fields may be omitted, but the quotation marks must be included to indicate missing fields, e.g., ´s1´´s3´.

The .CN control word allows the user to create index records which can be processed independently to form an index. Each index item is expanded to 40 characters (padded with blanks) and a blank is appended to the front of each record for a total record length of 121 characters.

After the source file has been scripted, the *2 file, containing the index records, may be sorted into ascending order using /SORT (see Chapter 3) with the following sort keys:

Key	Starting Column	Length
1	002	040
2	042	040
3	082	040

The sorted file can then be used as input to the SCRNDX program to create a "2-up" index for the particular document. SCRNDX is an interactive program which when run at a terminal will prompt the user for the information needed to create the index.

Notes:

(1) No individual item in the index data may be longer than 40 characters.

(2) If the user intends to use the SCRNDX program against the index data the index items should be used as follows:

"s1" - the major index item.
"s2" - the minor index item.
"s3" - the page number which can be generated using the special symbol "%" (see page 152).

(3) The index for this manual was generated using .CN control words and the SCRNDX program.

Example:

(1) The following line appears (just below the title for the previous page) in the script file that was used to create this manual:

```
.cn ´/SCRIPT control words´.CN´%´
```

The user is referred to the index of this manual to see it in its final form.

CONCATENATE cancels a previous NO CONCATENATE control word and causes output lines to be formed by concatenating input lines and truncating at the nearest word boundary to fit within the specified line length.

```
.CO
```

The CONCATENATE control word specifies that output lines are to be formed by shifting words to or from the next input line. The resulting line will be as close to the line length as is possible without exceeding it or splitting a word.

Defaults:

This command creates a break and is in effect until an .NC (or .NF) is encountered.

Notes:

(1) CONCATENATE is the normal mode.

(2) Output produced with CONCATENATE in effect is similar to what a normal typist produces (only if .NJ is in effect).

The CONDITIONAL PAGE control word causes a page eject to occur if less than a specified number of lines remain on the page.

```
| .CP          {n}                                                  |
```

The .CP control word will cause a page eject to occur if "n" lines do not remain on the current page.

This request is especially useful:

(1) Before a .SP control word which is used to leave room for a drawn figure. Use of .CP will guarantee that all of the spaces are on the same page. (Note that .LS YES must be specified if the spaces entered in this way are to be left on the next page.)

(2) Preceding a section heading to insure that the heading will not be left alone at the bottom of the page. In this case the .CP control word should follow the .SP control word which precedes the heading.

<u>Defaults:</u>

This command will not cause a break. If <u>n</u> is omitted the command will be ignored.

<u>Notes:</u>

(1) If no operand is specified, the .CP control word is ignored.

(2) By "remainder of current page" it is meant "the area between the last typed line and the beginning of saved footnote text (if any) or the current bottom margin (if no footnotes are saved)."

The DON´T COUNT control word causes all output produced by the immediately following text line to not be counted for purposes of page length control.

```
.DC
```

If run in the VPS batch, all output produced by the next text line will be overprinted on the last printed line.

If run at a terminal, output produced by the next text line will be typed normally, but will not be counted towards the page length.

The .DC control word is primarily useful to allow the user to store equations and such in a file when two or more typeballs are necessary during output.

Defaults:

This command will cause a break before processing the immediately following text line and it will be in effect only for that line. Subsequent lines will be processed normally.

Notes:

(1) Only text lines which immediately follow a .DC are not counted.

The DOUBLE SPACE control word causes a line to be skipped between each line of typed output.

```
.DS
```

Subsequent output lines will be double spaced.

Defaults:

This command is not in effect unless encountered. It does cause a break to occur when specified.

Notes:

(1) Single spacing is the normal mode.

(2) .SP control words encountered while .DS is in effect will produce twice the normal number of blank lines.

(3) Footnote lines are never double spaced.

(4) The operand to the .CP control word is not doubled when .DS is in effect.

The EMPTY PAGE control word is used to control suppression of empty (except for headings and footings) pages.

```
.EM        [ Yes ]
           [ No  ]
```

Empty pages (pages which would, if printed, contain only headings, footings, and possibly footnotes) can be generated in a number of ways and are _not_ normally printed by Script.

.EM YES specifies that empty pages _are_ to be printed. .EM NO specifies that empty pages are _not_ to be printed.

Defaults:

A break will not be created and empty pages will not be printed unless .EM YES or .EM is encountered.

Notes:

(1) If the operand is omitted, "YES" is assumed, since the user presumably intends to accomplish something with the control word.

(2) .EM NO is the normal mode.

(3) If pages are suppressed as a result of .EM NO, page numbers will still increment unless .PN OFFNO has been specified.

The ODD FOOTING control word specifies three items of title information to be printed at the bottom of odd-numbered pages.

```
| .FD        ⎡ s1         ⎤
|            ⎢ ´s1´s2´s3´ ⎥
|            ⎢ ´´´´       ⎥
|            ⎣ ____       ⎦
```

Where "s1", "s2" and "s3" are character strings <u>not</u> containing quotation marks. Any of the fields may be omitted, but the quotation marks must be included to indicate missing fields, e.g., ´s1´´s3´.

The .FD control word is used in a way similar to the .HE control word. The title items defined with .FD will be printed in a footing line near the bottom of odd-numbered pages.

The footing line is generated by:

(1) Substituting the current page number for each appearance of the special symbol "%".
(2) Left-adjusting "s1".
(3) Centering "s2".
(4) Right-adjusting "s3".

<u>Defaults:</u>

A break will not be created when this command is encountered. Unless otherwise specified, .FD ´´´´ will be in effect.

<u>Notes:</u>
(1) No individual item in the footing data may be longer than 60 characters.
(2) "s3" may overlay "s2" if necessary, and "s2" may overlay "s1" if necessary.
(3) The default value of the footing line is "´´´´´".

The FOOTING control word specifies three items of title information to be printed at the bottom of even- and odd-numbered pages.

```
 .FE        ┌ s1          ┐
            │ ´s1´s2´s3´  │
            │ ´´´´        │
            └ ____        ┘
```

Where "s1", "s2" and "s3" are character strings _not_ containing quotation marks. Any of the fields may be omitted, but the quotation marks must be included to indicate missing fields, e.g., ´s1´´S3´.

The .FE control word is used in a way similar to the .HE control word. The title items defined with .FE will be printed in a footing line near the bottom of even- and odd-numbered pages.

The footing line is generated by:

(1) Substituting the current page number for each appearance of the special symbol "%".
(2) Left-adjusting "s1".
(3) Centering "s2".
(4) Right-adjusting "s3".

Defaults:

A break will not be created when this command word is encountered. Unless otherwise specified, .FE ´´´´ will be in effect.

Notes:
(1) No individual item in the footing data may be longer than 60 characters.
(2) "s3" may overlay "s2" if necessary, and "s2" may overlay "s1" if necessary.
(3) The default value of the footing line is "´´´´".

The FOOTING MARGIN control word specifies the number of blank lines which are to be left between the bottom of formatted text and any footing line.

```
.FM        ⎡n⎤
           ⎣1⎦
```

The .FM control word defines the footing margin, which is the number of blank lines which will be left between the bottom of formatted text and any footing line on all pages.

If the footing margin is _n_ and the bottom margin is _m_, the bottom of each page will appear as follows:

(1) _n_ blank lines,
(2) The footing line (if any),
(3) _m_ - _n_ - 1 blank lines.

Defaults:

This command word creates a break when encountered and until then _n_ = 1 will be in effect. If the operand is omitted _n_ = 1 will be assumed.

The FOOTNOTE control word allows the user to enter a footnote which will be printed at the end of the current page.

```
.FN        {Begin}
           {End  }
```

When the .FN begin control word is encountered, the current values of all relevent control variables are saved and Script prepares to accept footnote text.

When the .FN end control word is encountered, the values saved by the .FN begin are restored and formatting of output continues.

Script saves enough room at the bottom of the page to print the footnotes and three separator lines. If enough room does not exist, footnotes will be continued at the bottom of the next page. Any control words (except .FN) may appear within the footnote. All page eject controls are ignored, space (.SP) requests in footnotes are in forced single space mode and all other requests apply <u>only</u> to the footnote. Format controls in effect when the .FN begin is encountered remain in effect during formatting of the footnote unless explicitly overridden. (E.g., if indentation of the text has been requested with an .IN control word, the footnote will also be indented.)

<u>Defaults</u>:

This command does not create a break.

<u>Notes</u>:

(1) Footnotes are always single spaced on output.
(2) No more than 100 formatted footnote lines may be waiting for output at any time.
(3) The .FN begin control does <u>not</u> act as a break. The next regular input line, that is, the first line after the .FN end, will be concatenated with the previous line.
(4) No footnotes will appear on a page if at least 4 lines are not available at the end of the page. If footnotes are generated within 5 lines of the bottom, they will be saved for the next page.
(5) The line which precedes a footnote definition will always be printed on the current page. It can never be displaced to the next page by accumulation of footnotes.
(6) Footnotes appear before the bottom margin set in the .bm control.

Example:

```
(a) As Dirac -1-
    .fn begin
    .in 5
    .un 5
    -1- Dirac, P.A.M.,
    The Principles of Quantum Mechanics, Oxford Press, 1958.
    .fn end
    has noted, this phenomenon is indigeneous to ...
```

Can be used to obtain output similar to that shown here:

```
(a) As  Dirac -1- has noted,  this phenomenon is  indigineous to
    ...
```

-1- Dirac, P.A.M., The Principles of Quantum Mechanics, Oxford Press, 1958.

The FORMAT control word combines the effect of CONCATENATE and JUSTIFY.

```
 .FO
```

The FORMAT control word is a short-hand way to specify CONCATENATE and JUSTIFY. This control word specifies that output lines are to be formed by shifting words to or from the next line (concatenate) and padded with extra blanks to produce an even right margin (justify).

Defaults:

This command does create a break. It is in effect unless a .NF is encountered.

Notes:

(1) Since FORMAT is the normal mode, the control word is used only to cancel a previous NO FORMAT control word.

The EVEN FOOTING control word specifies three items of title information to be printed at the bottom of even-numbered pages.

```
.FV       [ s1         ]
          [ ´s1´s2´s3´ ]
          [ ´´´´       ]
          [ ______     ]
```

Where "s1", "s2" and "s3" are character strings not containing quotation marks. Any of the fields may be omitted, but the quotation marks must be included to indicate missing fields, e.g., ´s1´´s3´.

The .FV control word is used in a way similar to the .HE control word. The title items defined with .FV will be printed in a footing line near the bottom of even-numbered pages.

The footing line is generated by:

(1) Substituting the current page number for each appearance of the special symbol "%".
(2) Left-adjusting "s1".
(3) Centering "s2".
(4) Right-adjusting "s3".

Defaults:

A break will not be created when this command is encountered. Unless otherwise specified, .FV ´´´´ will be in effect.

Notes:

(1) No individual item in the footing data may be longer than 60 characters.
(2) "s3" may overlay "s2" if necessary, and "s2" may overlay "s1" if necessary.
(3) The default value of the footer line is "´´´´".

The ODD HEADING control word is used to define three headings to be printed at the top of odd-numbered pages.

```
 .HD        [ sl          ]
            [ ´sl´s2´s3´  ]
            [ ´´´´        ]
            [ ____        ]
```

Where "sl", "s2" and "s3" are character strings <u>not</u> containing quotation marks. Any of the fields may be omitted, but the quotation marks must be included to indicate missing fields, e.g., ´sl´´s3´.

The .HD control word is used in a way similar to .HE. The headings defined by .HD are, however, printed only on odd-numbered pages (pages bound on the left margin).

The heading line is generated by:

(1) Substituting the current page number for each appearance of the special symbol "%".
(2) Left-adjusting "sl".
(3) Centering "s2".
(4) Right-adjusting "s3".

<u>Defaults:</u>

This command does not create a break when encountered. Unless otherwise specified, .HD ´´´PAGE %´ is will be in effect.

<u>Notes:</u>

(1) Appearance of .HD does not affect headings defined for even-numbered pages via .HE or .HV.
(2) See the description of .HE and .HV for further notes and examples.

The HEADING control word is used to define three headings to be printed at the top of both even- and odd-numbered pages.

```
.HE        ┌ s1         ┐
           │ ´s1´s2´s3´ │
           │ ´´´´       │
           │ ____       │
           └            ┘
```

Where "s1", "s2" and "s3" are character strings <u>not</u> containing quotation marks. Any of the fields may be omitted, but the quotation marks must be included to indicate missing fields, e.g., ´s1´´s3´.

The .HE control word specifies 3 items of character data to be printed near the top of all subsequent output pages.

The heading line is generated by:

(1) Substituting the current page number for each appearance of the special symbol "%".
(2) Left-adjusting "s1".
(3) Centering "s2".
(4) Right-adjusting "s3".

If the top margin is <u>m</u> and the heading margin is <u>n</u>, the top of the output page will consist of:

(1) <u>m</u> - <u>n</u> - 1 blank lines,
(2) the heading line, and
(3) <u>n</u> blank lines.

<u>Defaults:</u>

A break is not created by this command. Unless otherwise specified, .HE ´´´PAGE %´ will be in effect.

<u>Notes:</u>

(1) No individual item in the heading data may be longer than 60 characters.
(2) "s3" may overlay "s2" if necessary, and "s2" may overlay "s1" if necessary.
(3) The default value of the header line is "´´´PAGE %´".
(4) Appearance of the .HE control word overrides any previous values of the heading items. If no items are specified, no heading line at all will be generated. Furthermore, items which are omitted become null and do <u>not</u> retain their previous values.

.HE

Example:

```
    .he ´Department of Physics´´ATN-05-3-70-%´
```

The HEADING MARGIN control word specifies the number of blank lines which are to be left between the heading line and the first line of text.

.HM $\begin{bmatrix} n \\ \underline{1} \end{bmatrix}$

The heading line (if any) produced on subsequent output pages will be separated from the first line of text by $\underline{n}$ blank lines.

If the current top margin is $\underline{m}$ and the heading margin is $\underline{n}$, the top of each page will appear:

$\underline{m}$ - $\underline{n}$ - 1 blank lines
The heading line
$\underline{n}$ blank lines

Defaults:

This command creates a break and until it is encountered $\underline{n}$ = 1 will be in effect. If the operand is omitted $\underline{n}$ = 1 will be assumed.

Notes:

(1) The heading margin must always be strictly less than the top margin.

The EVEN HEADING control word is used to define three headings to be printed at the top of even-numbered pages.

```
.HV        ┌ s1          ┐
           │ ´s1´s2´s3´  │
           │ ´´´´        │
           └ ────        ┘
```

Where "s1", "s2" and "s3" are character strings not containing quotation marks. Any of the fields may be omitted, but the quotation marks must be included to indicate missing fields, e.g., ´s1´´s3´.

The .HV control word is used in a way similar to .HE. The headings defined by .HV are, however, printed only on even-numbered pages (pages bound on the right margin).

The heading line is generated by:

(1) Substituting the current page number for each appearance of the special symbol "%".
(2) Left-adjusting "s1".
(3) Centering "s2".
(4) Right-adjusting "s3".

Defaults:

This command does not create a break when encountered. Unless otherwise specified, .HV ´´´PAGE %´ is will be in effect.

Notes:

(1) Appearance of .HV does not affect headings defined for odd-numbered pages via .HE or .HD.
(2) See the description of .HE and .HD for further notes and examples.

The IF control word causes the next input line to be processed or not depending on the result of a comparison.

```
.IF       { ´s1´   [op]   ´s2´ }
          { n1            n2   }
```

Where:

"s1" and "s2" are character strings not containing blanks, "n1" and "n2" are signed decimal integers, and [op] is one of the following:

```
=     >
¬=    <=
<     >=
```

Note that for ASCII terminals on VPS the tilde, ~, should be used for the ¬.

If the comparison operation is true, the immediately following input line is processed normally. Otherwise, the next input line is skipped (ignored).

If the first operand begins with a quote (´), the comparison will be between two characters strings using the standard EBCDIC collating sequence.

If the first operand does not begin with a quote, the comparison will be between two signed decimal integers.

Defaults:

This command does not create a break when encountered.

Notes:

(1) If the two character strings to be compared have unequal lengths, the comparison will result in the shorter string being considered less than the longer.

Example:

(1)
```
.sr a % / 2 * 2 - %
.ur .if &a = 0
```
Will cause the next input line to be processed if the current page number is even.

The IMBED control word is used to insert the contents of a specified file into the printout of the file currently being formatted.

```
 .IM      ┌ indexname.filename ┐     ┌      *      ┐
          │ USERLIB.filename   │     │      -      │
          └ -------            ┘     │ nl      n2  │
                                     └ nl      *   ┘
```

The first argument names a file whose contents are to be included in the input. If an index name is not specified, USERLIB is assumed.

"nl" gives the item number of the first line of the imbedded file to be read, "n2" gives the item number of the last line to be read. Instead of "nl n2", the single character "*" may be used to indicate that the entire file is to be included. The "*" may also be used instead of "n2" to indicate that the rest of the file (that is, items "nl" through the end) are to be included.

The .IM and .AP control words perform similar functions, but .IM allows the contents of a second file to be inserted into the printout of an existing file rather than appended to it. Imbedding may be used to insert standard sets of control words at desired spots, or to control formatting of a long document out of individual files.

Files may be imbedded to 10 levels. Said in a different way, a file may imbed another file or itself and then that file may imbed another file or itself and then that file ... this can continue until the stack of imbedded files reaches 10.

Defaults:

This command word does not create a break when encountered. The entire file is read in starting at the first line, i.e., * is assumed.

Notes:

(1) /INCLUDE control statements (see the VPS Handbook) may not be used in script files. .AP (see page 100) or .IM should be used instead.

Examples:

(a) .im chapter4

The contents of the file named CHAPTER4 will be included in the input. When the end of CHAPTER4 is reached, printout of the current file will continue.

(b) .im footnote 12 15

Lines 12, 13, 14, and 15 of FOOTNOTE will be imbedded.

The INDENT control word causes the left margin of the printout to be indented a specified number of spaces.

```
.IN         [ n ]
            [ 0 ]
```

The .IN control word causes printout to be indented "n" spaces from the left margin. This indentation remains in effect for all subsequent lines (including new paragraphs, footnotes, and new pages) until another .IN is encountered. It will remain in effect even if .NF is specified.

.IN 0 or .IN will cancel the indentation and cause printout to continue at the original left margin.

Defaults:

When this command word is encountered it creates a break and until then n = 0 is in effect. When the operand is omitted n = 0 is assumed.

Notes:

(1) the .IN request causes any offset (.OF) setting or undent (.UN) setting to be cleared.

The JUSTIFY control word causes output lines to be padded with extra blanks so that the right margin is justified.

```
.JU
```

The .JU control word specifies that all subsequent output lines are to be formed by padding with extra blanks to cause the right margin to be justified.

Defaults:

This command creates a break and is in effect until an .NJ is encountered.

Notes:

(1) Since JUSTIFY is the normal mode, this control word is used to cancel a previous NO JUSTIFY control word or the NO JUSTIFY part of a NO FORMAT control word.

(2) If a line exceeds the current line length, and NO CONCATENATE is in effect, the line is printed as is.

(3) This control word is seldom used without CONCATENATE, which is the same as specifying FORMAT, which combines the two functions.

The LINE LENGTH control word specifies the number of horizontal character positions which are to be printed in subsequent output lines.

```
.LL        [ n ]
           [ 1 ]
```

The .LL control sets the length of subsequent output lines to "n" characters.

Defaults:

When this command word is encountered it creates a break. Unless otherwise specified _n_ = 60 characters per line (including blanks) will be in effect. If no argument is supplied then _n_ = 1 will be assumed.

Notes:

(1) Most terminals and printers print 10 characters per horizontal inch. The default value of 60 is sufficient to give 1 1/2 inch left and 1 inch right margins on 8 1/2 inch paper.

The LEADING SPACES control word is used to control suppression of blank lines at the beginning of a page.

```
 .LS        [Yes]
            [No ]
```

Blank lines may occasionally be the first lines of formatted output to be printed on a page. For example, a .SP 10 may have appeared 5 lines from the bottom of the previous page. Script does not print such blank lines except at the top of the first page of output.

Since blank lines may sometimes be desirable at the beginning of a page (e.g., when centering a figure on a page). the .LS control is provided.

.LS YES allows blank lines to begin a page. .LS NO prevents blank lines from beginning a page.

Defaults:

This command does not create a break and .LS NO is in effect until .LS YES or .LS is specified. (If the operand is omitted, "YES" is assumed.)

Notes:

(1) If the operand is omitted, "YES" is assumed.
(2) .LS NO is the normal mode.
(3) Leading spaces are allowed on the first page of printed output.
(4) This discussion applies only to formatted text, not to headings or top margins.

The NO CONCATENATE control word stops words from shifting to or from the next line.

```
.NC
```

The NO CONCATENATE control word stops words from shifting to and from the next line. There will be a one-to-one correspondence between the words in input and output lines.

Defaults:

This command word does create a break and is not in effect until encountered.

Notes:

(1) Concatenation will be completed for the lines which precede the .NC control word, since it also acts as a BREAK.

(2) If JUSTIFY is in effect, the right margin will still be adjusted.

The NO FORMAT control word causes lines to be typed as they appear in the input by negating the effect of JUSTIFY and CONCATENATE.

```
.NF
```

The .NF control word is a short-hand way to specify .NJ and .NC. Subsequent output lines will by typed exactly as they appear in the input until a .FO, .JU, or .CO control word in encountered.

Tables and equations are more easily entered if .NF is used to suppress formatting just before them.

Defaults:

This command does create a break and is not in effect until encountered.

Notes:

(1) The diagnostic "LINE TOO LONG OR PRINTER ERROR" is commonly caused by failing to specify .FO after a section produced under .NF.

The NO JUSTIFY control word stops justification of the right margin.

```
 .NJ
```

The .NJ control word causes Script to cease inserting extra spaces into lines in order to justify the right margin. If CONCATENATE is in effect, the output lines will be as long as possible without exceeding the current line length and without breaking words in the middle. The resulting output will be similar to what an ordinary typist would produce.

<u>Defaults</u>:

This command does create a break and is not in effect until encountered.

The OFFSET control word causes all but the first line of a section to be indented.

```
.OF        [n]
           [0]
```

The .OF control word is used to indent the left side of the printout "n" spaces after printing the next output line. The indentation remains in effect until a BREAK or until another .OF control word is encountered.

The .OF control word may be used within a section which is also indented with the .IN control word. Any subsequent .IN control word causes the indentation set by .OF to be cleared.

If it is desired to start a new section with the same offset as the previous section, it is necessary to repeat the .OF request.

Defaults:

This command creates a break when encountered. If no argument is supplied then n = 0 will be assumed.

Notes:

(1) If the following sequence is input:

```
.of m
<text>
.of n
<text>
```

The second offset will be added to the first.

Examples:

(1) Many of the sections of this manual (including the example text) are produced by a sequence like this:

```
<section header>
.in 5
.of 4
(1) <text>
.of 4
(2) <text>
   .
   .
   .
.in
```

The PAGE control word causes an output page eject.

```
| .PA          [n]                                                  |
```

When the .PA control word is encountered, the rest of the current page is skipped, any saved footnote lines are printed, the footer line is printed, and a new page is begun whose number is "n".

Defaults:

This command word does create a break when encountered. If the operand is omitted then the current page number plus one (% + 1) is assumed.

Notes:

(1) If the STOP option was specified in the control line which invoked Script, output will cease at the end of the current page to allow the typist to insert the next page of paper.

(2) If the current page is empty and .EM NO is in effect, either by default or through explicit statement, the .PA control has no effect other than causing the page number to be incremented or reset. Consequently, heading lines are not printed until there is a text line about to be printed which allows dynamic changes to page headings and possibly remote lines imbedded in the text of that page.

(3) Heading lines, margins, etc., must be specified before the first line of text which will appear on the new page.

(4) If no argument is supplied, the page number will be incremented by 1.

The ODD PAGE control word causes output to continue on an odd-numbered page.

```
.PD
```

The .PD control word causes one or two page ejects so that output continues on an odd-numbered page.

Defaults:

This command will create a break when encountered.

Notes:

(1) If empty-page suppression is on (.EM NO), the page number will be incremented to the next odd page and output will continue on the next physical page.

The PAGE LENGTH control word specifies the physical size of the output page in units of typewriter lines.

```
.PL        [n]
           [1]
```

The .PL control word allows the use of various paper sizes for output.

Normal 8 1/2 x 11 inch paper is 66 lines long on a IBM 1403 printer, a IBM 2741 type of terminal, or any other device which types 6 lines per vertical inch.

Defaults:

This command word will create a break and unless otherwise specified n = 66 will be in effect. If the command is encountered and the operand is omitted n = 1 will be assumed.

Notes:

(1) Use of the .PL control word for any other purpose than specifying the actual physical size of the output page is discouraged. The TOP MARGIN and BOTTOM MARGIN control words should be used to control the dimensions of printed text.

The PAGE NUMBER control allows the user to control the incrementing of page numbers.

```
.PN    ┌ON    ┐
       │OFF   │
       └OFFNO ┘
```

OFFNO	Suppresses incrementing of page numbers.
OFF	Suppresses the printing of page numbers.
ON	Causes incrementing and printing of page numbers to resume.

The .PN control word is used to control the automatic incrementing of page numbers. If OFFNO is specified, page numbers will not increase as subsequent pages are output. If ON is specified, incrementing and printing will resume.

Defaults:

This command word will not create a break. Unless otherwise specified "ON" will be in effect.

Notes:

(1) If page incrementing is suppressed, the even and odd page force control words (.PV and .PD) function exactly like .PA.

The PRINT control word causes one line of information to be typed on the user´s terminal.

```
.PR        [information]
```

If output is to the user´s terminal, the .PR control is ignored (except during the first pass of a 2PASS run, when .PR causes output normally). Otherwise, the entire operand field is printed on the terminal.

The .PR control may be of use, for example, immediately preceding a .RC (Read Control) control word as a reminder of what to do.

Defaults:

This command word will not create a break when encountered.

Example:

```
.pr Type Name and Address (3 lines)
.rc
.br
.rc
.br
.rc
.br
```

The EVEN PAGE control word causes output to continue on an even-numbered page.

```
 ___________________________________________________________________
|                                                                   |
|  .PV                                                              |
|___________________________________________________________________|
```

The .PV control word causes one or two page ejects so that output continues on an even-numbered page.

Defaults:

This command word will create a break when encountered.

Notes:

(1) If empty-page suppression is on (.EM NO), the page number will be incremented to the next even page and output will continue on the next physical page.

The RIGHT ADJUST control word causes the next input line to be right adjusted in the output line.

```
.RA
```

The next input text line, including any leading or trailing blanks, will be right adjusted in the output line.

Defaults:

This command word will create a break when encountered.

Notes:

(1) If the next line is longer than the current line length, it will be truncated.

Example:

(1) .ra
(1.5)
Produces:

```
                                                        (1.5)
```

The READ CONTROL control word allows the user to enter control or input lines during processing of the input file.

```
.RC        [n]
           [1]
```

When the .RC control word is encountered the user's terminal keyboard is unlocked and n lines are accepted and processed as if they had been in the input file. The lines thus read may be prose or control information.

Defaults:

The .RC control word does not act as a break in itself. However, control words read under its control may act as breaks. If the operand is omitted then n = 1 is assumed.

Notes:

(1) A completely null line is taken to mean that the user is done typing input lines. The .RC is terminated regardless of whether n lines have been read. A null line signals an End Of File (EOF).
(2) The user might wish to use .RC in order to dynamically specify control words or prose.

Example:

```
.cm user will enter name and address here
.rc 4
```

The READ control word allows the user to type a line on the output page during Script output.

```
.RD          [n]
             [1]
```

When the .RD control word is encountered during output to the user's terminal, Script will unlock the keyboard until "n" carriage returns have been typed. The lines typed are ignored completely except for purposes of counting lines on the current page.

When the .RD control word is encountered during output to the offline printer, Script will insert n blank lines in the output.

This control word may be useful to allow addresses to be inserted in form letters or to allow the user to change type elements.

Defaults:

This command word does create a break when encountered. If the operand is omitted n = 1 is assumed.

Notes:

(1) If output is offline and the .RD is received within n lines of the bottom margin and .LS NO is in effect, spaces will not appear at the top of the next page. If, however, output is online and the other two conditions are met, spaces will appear at the top of the next page.

The REMOTE control word allows the user to save one or more input lines, which will be automatically imbedded at a specific place on the current page, the next page, or subsequent pages.

```
.RM        [n]        ⎡SAVE   ⎤
                      ⎢NOSAVE ⎥
                      ⎣DELETE ⎦
```

Where:

<u>n</u> is a decimal integer in the range:
TOP MARGIN < <u>n</u> < BOTTOM MARGIN

The input lines between the first .RM and the next .RM are saved. When the next line <u>n</u> is to be printed, the saved lines are automatically interpreted.

If NOSAVE is specified or assumed by default, the saved lines are erased after the first use.

If SAVE is specified, the remote sequence is saved and invoked on line <u>n</u> of every page until a .RM n DELETE is received specifying the same line number.

The .RM control word is useful for defining multi-line headers or for causing automatic insertion of figures.

Following the imbedding of a remote sequence, line formatting options are restored to their values prior to the insertion of the remote.

<u>Defaults</u>:

This command word will not create a break when encountered. If both operands are omitted the first operand will be set to the TOP MARGIN plus one, i.e. (<u>n</u> = TM + 1), and the second operand will become "NOSAVE". If only the first operand is specified, the second will become "NOSAVE".

<u>Notes</u>:

(1) <u>Any</u> lines whatever may appear within the .RM (except another .RM) and are interpreted when the remote sequence is inserted.

(2) The combined total number of footnote and remote lines may not exceed 100.

(3) Remote sequences are "triggered" when line _n_ of the output page is about to be printed. While a remote sequence is being used for input, no other remote may be triggered. If two or more remotes specify the same line number, only the oldest will be triggered.

(4) A .RM DELETE deletes the oldest sequence for the specified line number.

(5) If no arguments are presented, a line number one bigger than the current top margin is assumed.

Example:

(1) Assume a TOP MARGIN of 6. The following sequence places a figure at the top of the next page:

```
.rm 7
.in
   (Figure)
.ce
FIGURE ONE: Water Concentration in Lake Erie as
.ce
a Function of GNP.
.rm
```

The ROMAN control word causes page numbers to use lower-case roman numerals.

```
.RO
```

The .RO control word causes all page numbers produced hereafter in headings or footings to be printed in lower-case roman numerals.

Defaults:

This command word will not create a break and unless encountered will not be in effect.

The SPACE control word generates a specified number of blank lines before the next typed line.

```
.SP        [n]
           [1]
```

The SPACE control word causes n blank lines to be typed. If the end of the page is reached before satisfying the request, the remaining spaces may or may not be typed on the next page depending on whether .LS YES has been specified. If DOUBLE SPACE or TRIPLE SPACE is in effect, twice the specified number of spaces are typed.

Defaults:

This command does create a break when encountered and if the operand is omitted n = 1 will be assumed.

Notes:

(1) .SP control words within footnotes cause blank lines to be generated within the printout of the footnote at the bottom of the page.

The SET REFERENCE control word allows the user to assign a character or numeric value to a symbolic reference name.

```
.SR        name        { 'string'          [s]        [l] }
                       { numeric-expression               }
```

Where "s" is the starting column of the substring to be assigned (the default is 1) and "l" is the length of the substring (the default is = length of substring - s + 1).

The reference name named by the first operand is given the value of the second operand. If the reference name does not exist, it will be created.

The .SR control word may be used for a variety of purposes. For example:

(1) Symbols may be defined as having as their value the number of the page on which they appear and can be used to construct a table of contents.
(2) Symbols can be assigned to count the number of equations being used and to number the equations as they appear on the output.

The reference name may be any character string of length 7 or fewer which does not contain a blank or period. The name is converted to upper case prior to use.

The character string may include blanks and apostrophes within the string must be doubled.

The form of the numeric-expression which may be used as the assigned value is as follows:

```
<expression> =:: <term> | <expression> <op> <term>
<term> =:: <symbol> | <decimal-integer>
<op> =:: + | - | * | /
```

At least one blank must separate all items in the expression. That is, "5 + % - 7" is valid, while "5+%-7" is not. One unary "+" or "-" is allowed to immediately precede the number (with no intervening blanks). Operators are performed left to right.

The following special <symbol>'s are recognized:

(1)	"%"	Current page number.
(2)	"<*"	Current line number of file.
(3)	"<l"	Current line number of page.

Defaults:

This command word will not create a break when encountered. If one or both operands are omitted it will be treated as an error.

Notes:

(1) No more than 200 reference names may be defined during one run of Script.

(2) See the description of the .UR control word and the 2PASS option for further hints on the use of .SR.

Example:

(1) The sequence:

```
.sr eqnno 0
   .
   .
   .
.ur .sr eqnno &eqnno + 1
.ur .sr xeqn &eqnno
.ur x = erfc(a - bc)                              (&eqnno.)
.ur .sr eqnno &eqnno + 1
.ur y = erfc(a + bc)                              (&eqnno.)
   .
   .
   .
.ur Using the previous result for x (Equation &xeqn.)...
```

Produces:

```
x = erfc(a - bc)                          (1)
y = erfc(a + bc)                          (2)
Using the previous result for x (Equation 1)...
```

(2) The sequence:

```
.sr a 0
.sr b -5
.sr c 'a'
.sr d 'b'
.ur .ur .sr &c &&&d + 1
.ur a = &a..
.ur b = &b..
```

Produces:

```
a = -4.
b = -5.
```

(3) The sequence:

```
.sr page %
.ur The number of this page is &page..
```

Produces:

The number of this page is 154.

(4) The sequence:

```
.sr value 16
.ur .sr v2 ´&value          ´ 1 5
.ur  ***&v2.***
```

Produces:

```
***16   ***
```

The SINGLE SPACE control word causes output to be single spaced.

```
.SS
```

Subsequent output lines will be single spaced.

Defaults:

When this command word is encountered it will create a break. Unless otherwise specified .SS will be in effect.

Notes:

(1) Since SINGLE SPACE is the normal mode, this control word is used to undo the effect of a previous .DS or .TS.

The TAB SETTING control word specifies the logical tab locations Script is to use when the logical tab character (see also the .TC control word) is found in the following text lines.

```
 .TB   t1 t2 ...tn
```

A tab character present in the text is replaced with one or more blanks to simulate the effect of physical tabs. The .TB control word specifies the locations of the logical tabs.

A .TB control word with no operands causes reversion to the default tab settings.

<u>Defaults</u>:

The control word creates a break when encountered. Until .TB is found with operands the logical tab locations are columns 5, 10, 15, ..., 80.

The TAB CHARACTER control word allows a character to be specified as the Script logical tab character.

```
.TC    [character]
```

A tab character present in the text is replaced with one or more blanks up to the next logical tab location (see the .TB control word).

A .TC control word with no operands resets the logical tab character to the default.

Defaults:

Unless otherwise specified the logical tab character is the EBCDIC tab character (hexadecimal 05).

Notes:

(1) The tab character is never printed in the formatted output listing.

Examples:

(1) The sequence:

```
.tc
 4 12
```

Produces:

```
    4    12
```

The TOP MARGIN control word specifies the number of lines which are to be placed between the physical top of the output page and the first line of text.

```
.TM        [ m ]
           [ 1 ]
```

Subsequent output pages will begin with <u>m</u> lines (which may include a heading line) before the first line of text.

<u>Defaults</u>:

This command word will create a break and until encountered <u>m</u> = 5 will be in effect. When encountered without an operand <u>m</u> = 1 will be assumed. This may result in an error.

<u>Notes</u>:

(1) The TOP MARGIN must always be strictly greater than the HEADING MARGIN.

(2) The margin specified by .TM will apply to the current page if no output has yet been produced on it.

The TRANSLATE control word allows the user to specify the contents of the translate table used to implement the TRANSLATE option.

```
.TR        s1 t1 [s2 t2 ...]
```

Where "s1", "t1", etc., are single characters or two-digit hexadecimal numbers using upper or lower case letters.

If the TRANSLATE option was specified in the command line which invoked Script, all subsequent output lines will be printed with all occurrences of "s1" replaced by "t1", etc.

The .TR control word is primarily of use when output must use a different character set than was used to create the file. For example, the user may print on line a file which uses special characters not available on the terminal (e.g., the TN train's superscript characters are not available on the 2741 terminal).

Defaults:

This command will not create a break when encountered. Lower case to upper case translation will be in effect until .TR is encountered with operands.

Notes:

(1) Heading, footing, and footnote lines are translated under control of the translate table current when the line is output.

(2) Script control lines are never translated.

(3) Translate pairs remain active until explicitly re-specified.

(4) The default translate table takes lower case letters into upper case letters and leaves all other characters the same.

(5) Hexadecimal numbers are recognized by the presence of 2 characters (instead of one) and may use upper or lower case letters A-F.

(6) The last pair in a .TR line may consist of only one argument, which indicates that the corresponding character is to be translated into itself (left unchanged).

Examples:

(1) .tr 8D (9D) B0 0 ... B9 9
Causes the TN train´s superscript parentheses and numbers to print as ordinary parentheses and numbers.

(2) .tr 40 ?
Causes all blanks in the file to be typed as "?" on output.

(3) .tr 40 ?
Causes all blanks in the file to be typed as "?" on output, and the " " to appear as " " (i.e., to be translated into itself.)

The TRIPLE SPACE control word causes two lines to be skipped between each line of typed output.

```
.TS
```

Subsequent output lines will be triple spaced.

Defaults:

This control word is not in effect unless encountered. It causes a break to occur when specified.

Notes:

(1) Single spacing is the normal mode.

(2) .SP control words encountered while .TS is in effect will produce twice the normal number of blank lines.

(3) Footnote lines are never triple spaced.

(4) The operand to the .CP control word is not tripled when .TS is in effect.

The UNDERLINE control word allows a character to be specified as the underline character.

```
.UL        [character]
```

After the underline character is specified, the presence of an underline character in the text will cause Script to underline <u>all</u> characters (excluding blanks) until the next occurrence of <u>the</u> underline character. If blanks are to be underlined the Back Space character should be used.

<u>Defaults</u>:

Unless otherwise specified, no underline character is defined.

<u>Notes</u>:

(1) See also the description of the .BS (Back Space) control word (page 105).

(2) It may be necessary to increase the terminal linesize value when doing heavy underlining with the .UL character. The VPS /CTL command (see the VPS Handbook) should be used. The following would set the linesize value to 250 characters:

```
/ctl linesize 250
```

<u>Example</u>:

(1) The sequence:

```
.ul
This is an  example of the underline function
```

Produces:

This is an <u>example</u> <u>of</u> <u>the</u> <u>underline</u> <u>function</u>

The UNDENT control word forces the next output line (only) to start a specified number of columns to the left of the current indent.

```
.UN          [n]
             [0]
```

The .UN control word causes only the next output line to begin _n_ spaces further left than the current indentation.

The .UN control word serves a similar purpose as .OF, but in a different way. The choice between the two is largely a matter of personal preference.

Defaults:

This command will create a break when encountered. If the operand is omitted then _n_ = 0 will be assumed.

Notes:

(1) The undentation may not exceed the current indentation.

Examples:

(1) The following two sequences are equivalent:

```
.in 5
.of 4
<text>

.in 9
.un 4
<text>
```

The USE REFERENCE control word causes an input line to be reformatted by substituting the current values of specified reference names in the line and to be processed as if it had been in the input.

```
.UR          line
```

The line which begins one blank after the .UR control word is reformatted by replacing occurrences of strings of the form "&name" with the current value of the reference name called "name".

A reference name instance is denoted by the appearance of a "&" followed by the name of the desired reference name followed by either a blank or a period. The current value of the reference name is converted (if necessary) to a character string and substituted for the string "&name." (if a period is used for delimiting) or the string "&name" (if a blank is used).

The line thus constructed is processed as if it had been in the original input stream.

Defaults:

The .UR control word does not act as a break itself. However, control words within the line may create a break when the line is reinterpreted after substituting symbolic references with their values. If line is omitted a blank line is assumed.

Notes:

(1) See the description of the .SR control word for additional information.

(2) If reference names appearing in the line have not yet been assigned a value via .SR, a null character string value will be assumed. Note, however, the effect of the "2PASS" option previously described. Note also that such reference names do not count toward the total of 200.

(3) Text &'s in a .UR line are indicated by "&&". Only one "&" appears in the reformatted line.

(4) Negative values are allowed for numeric reference names and result in a signed integer suitable for use with the .SR control word.

SAMPLE INPUT TO SCRIPT

```
.ul
.ce
SAMPLE OUTPUT FROM SCRIPT
.rm 45
.ra
EXAMPLE OF REMOTE CONTROL
.rm
.sp 2
"Oh nuts!  I have to type this whole page over".  That
complaint is probably heard many times in the course of
preparing manuscripts--reports, letters, minutes,and so on.
Then there is the problem of making duplicate  original
copies of a typed
manuscript.  When we use an ordinary typewriter,
text-processing, as this is called, can be a time-consuming
and harrassing job.  The Script facility, which operates
under VPS(*),
.fn begin
(*) Virtual Processor System
.fn end
was written to handle these procedures for you;
you merely type in the first version and make
any necessary corrections.  By
using the "commands" that you type in with your lines of
text, the computer prints out as many "first" copies as you
wish, with margins, spacing, indentation, etc., performed
in accordance with your commands.
.sp 1
One of the most useful features of Script is right-margin
justification, as in book and newspaper printing.  This
means that your text is spaced as evenly as possible
between the margins of your printed page, filling in with
blanks where  necessary.  The printed line is
under control of a "line-length" command; any
short line that you type in will "grab" words
from a longer line above or below it, and fill out the
line with blanks where necessary to avoid splitting words
at the margins.  This is the ordinary (or default) mode
under Script.  You can, however, have lines printed out
exactly as you type them in, with no justification
performed by Script, by including a command that instructs
the computer to turn off the format mode.  This is useful
for typing figures and charts.  Line justification can
later be respecified if desired.
.cm Last line of Input Text.
```

SAMPLE OUTPUT FROM SCRIPT

"Oh nuts! I have to type this whole page over". That complaint is probably heard many times in the course of preparing manuscripts--reports, letters, minutes,and so on. Then there is the problem of making duplicate <u>original</u> copies of a typed manuscript. When we use an ordinary typewriter, text-processing, as this is called, can be a time-consuming and harrassing job. The Script facility, which operates under VPS(*), was written to handle these procedures for you; you merely type in the first version and make any necessary corrections. By using the "commands" that you type in with your lines of text, the computer prints out as many "first" copies as you wish, with margins, spacing, indentation, etc., performed in accordance with your commands.

One of the most useful features of Script is right-margin justification, as in book and newspaper printing. This means that your text is spaced as evenly as possible between the margins of your printed page, filling in with blanks where necessary. The printed line is under control of a "line-length" command; any short line that you type in will "grab" words from a longer line above or below it, and fill out the line with blanks where necessary to avoid splitting words at the margins. This is the ordinary (or default) mode under Script. You can, however, have lines printed out exactly as you type them in, with no justification performed by Script, by including a command that instructs the computer to turn off the format mode. This is useful for typing figures and charts. Line justification can later be respecified if desired.

EXAMPLE OF REMOTE CONTROL

(*) Virtual Processor System

Chapter 3: /SORT - Library File Sorting

The /SORT command invokes an interactive program which will sort one or more library files into ascending or descending sequence and produce a single sorted output file. The sort is performed in core and therefore the capacity is limited by the amount of virtual storage available. Chapter 9 describes the OS SORT/MERGE package which can be used to sort large library files as well as tape files and work files. The /SORT command may be used only at the terminal. The format of the /SORT command is shown below.

```
/SORT    filename,filename,...
```

filename
: specifies the library file(s) to be sorted. It may be specified as a single library file name or as a list of file names separated by commas.

Sort Keys

After the /SORT command has been entered on the terminal the program responds with file statistics (i.e., number of files, number of records, and record length) and then prompts the user for sorting information with the following message:

```
*ENTER SORT KEYS
```

At this time the user may enter up to ten sort keys (sorting fields) which will control the action of the sort. Sort keys have the following format:

```
sssnnno
```

Where:

- sss - is the starting column of the sort field on each record (the first column being column 1). Note that leading zeroes must be typed.
- nnn - is the length of the sort field in characters (bytes). Again, leading zeroes must be typed.
- o - identifies the sorting order for this field and is either an "a" indicating ascending order or a "d" indicating descending order.

Sort Keys are entered starting in the first position of a line and there may be only one sort key per line. If more than one sort key is specified, the first sort key is the major sorting field followed by the other sort keys - minor sorting fields. /SORT processes the records by the order of the sort keys specified (i.e., all the records are sorted as specified by the first sort key; then, within that breakdown records will be sorted as specified by the second sort key; and so on). After each sort key is entered, /SORT will respond with a message defining the key.

To terminate the entering of sort keys, the user types the following starting in the first position of the line:

```
/end
```

Saving the Sorted Output File

After the sort has completed /SORT will prompt the user with the following message:

```
*ENTER SAVE OR RSAV COMMAND
```

At this time the user may enter the keyword SAVE followed by a file name if the sorted output file is to be saved as a new VPS library file; or the keyword RSAV followed by a file name if the sorted output file is to replace an existing library file.

/SORT Example

In the following example the files TRICYCLE and SCOOTER are to be sorted using two sort keys. The major sort key starts in column 10 and has a length of 8 bytes. For this sort key the records are to be sorted in ascending order. The minor sort key starts in column 35 and is one byte in length. For this sort key the order is descending. The sorted output file replaces the file TOYS.

```
/sort tricycle,scooter
STATISTICS FOR SORTING:
NUMBER OF FILES   ...............2
NUMBER OF RECORDS .............342
RECORD LENGTH IS  .............121
ENTER SORT KEYS
010008a
KEY 01:COLUMN 010,LENGTH 008,ORDER A
035001d
KEY 02:COLUMN 035,LENGTH 001,ORDER D
/end
ENTER SAVE OR RSAV COMMAND
rsav toys
*GO
.
.
.
```

Chapter 4: COPY - File Copying

The COPY program is used to copy data from one file (i/o unit) to another. COPY reads control statements from either the input stream or the keyboard: the input stream is accessed first, and the keyboard is read when end-of-data is reached on the input stream (if COPY is being used on the batch, end-of-data on the input stream causes job termination). A COPY control statement consists of a series of keywords separated by one or more blanks. Each statement must indicate which i/o unit is to be used for input (the IN keyword) and which for output (the OUT keyword). A statement is not considered complete until both IN and OUT have been specified - if the statement is being entered through the keyboard and return is hit before both these parameters have been defined, the user will be polled for more information.

The following keywords are accepted by COPY:

```
 {IN     n     }  {OUT    n     }   [QUIT]  [CLEAR]   [LRECL n]
 {       ddname}  {       ddname}

[PAGE  ]    [CC  ]    [EOF  ]    [DEFINE filename]
[NOPAGE]    [NOCC]    [NOEOF]

[SAVE  filename]     [INFILNO n]     [OUTFILNO n]
[RSAV          ]
[REPL          ]

[INBLK n]     [OUTBLK n]     [INLRECL n]     [OUTLRECL n]

[INRECFM recfm]     [OUTRECFM recfm]
```

IN

IN is a required parameter which specifies which unit is to serve as input for the copy operation. The unit may be given either as a unit number or as a ddname.

OUT

OUT is a required parameter which specifies which unit is to receive the output during the copy operation. The unit may be given either as a unit number or as a ddname.

QUIT

QUIT causes immediate termination of the COPY program.

CLEAR

CLEAR resets all COPY parameter settings to their default values. This is useful if a statement has been partially entered and the user wishes to reissue it with different parameters. Any parameters appearing after CLEAR on the same physical input line will be lost.

LRECL

LRECL is used to specify the record length of records written to the OUT unit. If LRECL is less than the record length of the IN unit, the records will be truncated. If LRECL exceeds the IN unit record length, the records will be padded on the right with blanks. The default LRECL is the record length of the input unit. Note that LRECL will not affect the actual record length of DISK or TAPE output files, but it will affect the record length of library output (LIBOUT) files. The record length of a DISK output file is determined by the /FILE statement for that file, and the record length of a TAPE output file will also be taken from the /FILE statement unless it is overridden with the OUTLRECL parameter (see below).

PAGE | NOPAGE

If PAGE is specified, page headers will be included in the output being sent to the OUT unit. NOPAGE will suppress this function. If the OUT unit is of type TERMOUT or PRINTER, PAGE is the default. NOPAGE is the default for all other output unit types.

CC | NOCC

If CC is specified, a blank carriage control character will be inserted in front of each record before it is sent to the OUT unit. NOCC suppresses this action. If the OUT unit is of type TERMOUT or PRINTER, CC is the default. NOCC is the default for all other output unit types.

EOF | NOEOF

If EOF is specified, a write-end-of-file operation will be done on the output unit when the copy is completed. No such operation will be performed if NOEOF is given. If the output unit is of type DISK or TAPE, EOF is the default. The default for all other output unit types is NOEOF.

DEFINE

DEFINE indicates that COPY is to redefine the file being read by the library input (LIBIN) unit specified by the IN parameter (a DFLIBIN operation is performed - see the VPS System Facilities for Assembler Programmers manual). If the IN unit is not of type LIBIN, the DEFINE parameter is ignored.

SAVE | RSAV | REPL

Use of one of these parameters causes COPY to save the output in the library (SAVE) or to replace an existing library file with the output (RSAV, REPL). If the OUT unit is not of type LIBOUT, these parameters are ignored. RSAV and REPL are synonyms.

INFILNO

INFILNO applies only to TAPE input files and causes the file number given in the LABEL parameter of the /FILE statement to be redefined (see the VPS Handbook). If the IN file is not a TAPE file, INFILNO is ignored.

OUTFILNO

OUTFILNO applies only to TAPE output files and causes the file number given in the LABEL parameter of the /FILE statement to be redefined (see the VPS Handbook). If the OUT file is not a TAPE file, OUTFILNO is ignored.

INBLK

INBLK is used to redefine the blocksize for a TAPE input file.

OUTBLK

OUTBLK redefines the blocksize for a TAPE output file.

INLRECL

INLRECL redefines the logical record length for a TAPE input file.

OUTLRECL

OUTLRECL redefines the logical record length for a TAPE output file.

INRECFM

INRECFM allows the user to redefine the record format of a TAPE input file. The record format is given in the same format as it would be given in the RECFM parameter of a /FILE statement (see the VPS Handbook) - e.g. F, VB, etc.

OUTRECFM

OUTRECFM is used to redefine the record format of a TAPE output file. As for INRECFM, the record format is specified in the same format as it would be given in the RECFM parameter of a /FILE statement.

Examples of File Copying

In the first example, we wish to save the first two files on tape unit 1 in the library as TPFIL1 and TPFIL2. To do this, we run the job (see the VPS Handbook for a description of VPS job structure):

```
/FILE UNIT=(1,TAPE),LABEL=(1,NL),VOL=SER=TAPE00,
/     BLKSIZE=800,LRECL=80,RECFM=FB
/LOAD COPY
```

where we have defined unit 1 as the first file on the tape. We assume that the second file has blocksize 1320 and record length 132. In this and the other examples, messages typed out by COPY are shown in upper case, and lines typed in by the user are shown in lower case. The conversation appears as follows:

```
ENTER :in 1 out 10 save tpfil1

OUTPUT LIBRARY FILE USERLIB.TPFIL1: SAVED

265 RECORDS COPIED FROM UNIT 1 TO UNIT 10
ENTER :inblk 1320 inlrecl 132 infilno 2
MORE :in 1 out 10 save tpfil2

OUTPUT LIBRARY FILE USERLIB.TPFIL2: SAVED

430 RECORDS COPIED FROM UNIT 1 TO UNIT 10
ENTER :quit
*GO
```

Note that the LIBOUT unit used to save the files was unit 10, which always exists unless it has been specifically overridden by a /FILE statement or suppressed by a TRANS parameter (see the VPS Handbook). Naturally, a LIBOUT unit could have been specifically defined with a /FILE statement and used instead.

In the next example, we write library file DATA6 onto the TAPE unit with ddname ARCHIVE as physical file 6 and then, by suppressing the EOF function, write the two library files STUFF98 and STUFF99 onto the tape as file 7. The job we run is:

```
/FILE UNIT=(TAPE,NAME=ARCHIVE),LABEL=(,NL),VOL=SER=ARCHTP
/FILE UNIT=(LIBIN,NAME=LIBUNIT)
/LOAD COPY
```

We wish to write all the records out with blocksize 4000, record length 100, and record format FB. The conversation runs as follows:

```
ENTER :outblk 4000 outlrecl 100 outrecfm fb out archive
MORE :outfilno 6
MORE :in libunit define data6

598 RECORDS COPIED FROM UNIT LIBUNIT TO UNIT ARCHIVE
ENTER :outfilno 7 out archive noeof in libunit define stuff98
```

```
450 RECORDS COPIED FROM UNIT LIBUNIT TO UNIT ARCHIVE
ENTER :out archive in libunit define stuff99

231 RECORDS COPIED FROM UNIT LIBUNIT TO UNIT ARCHIVE
ENTER :quit
*GO
```

In the following example, we print the contents of the DISK file defined as unit 2. We use the standard TERMOUT unit 6 for the output and suppress the page headers. The job is:

```
/FILE UNIT=(2,DISK),VOL=SER=SYSFIL,DSN=UR.XYZ100.FILE,
/     DISP=OLD
/LOAD COPY
```

and the conversation takes the form:

```
ENTER :in 2 out 6 nopage
LINE 1
LINE 2
    .
    .
    .
LINE 50

50 RECORDS COPIED FROM UNIT 2 TO UNIT 6
ENTER :quit
*GO
```

Chapter 5: LKED - the Linkage Editor

The primary purpose of the linkage editor is to combine and link object modules (decks) into a load module. In the process of creating the load module all references between control sections are resolved as though they were assembled or compiled as one module. The load module produced by the linkage editor consists of executable, machine-language code in a format that can be loaded into virtual storage and relocated by VPS. Unlike the loader, which performs many of the same functions, the load module produced by the linkage editor is placed in a work file for later execution using the /LOAD statement. Object modules must be processed in this manner each time they are loaded for execution. For programs which are executed often and modified infrequently, the linkage editor provides a means of alleviating much of the redundant processing associated with the loading process.

In addition to combining and linking the object modules, the linkage editor performs the following functions:

- Library calls. Program load libraries (such as those containing standard subroutines) can be automatically searched to resolve external references. If unresolved external references remain after all the input to the linkage editor is processed, the linkage editor will search the automatic call libraries to resolve the references.

- Program modification. Control sections can be replaced, deleted, or rearranged (in overlay programs) during the linkage editor process, as directed by linkage editor control statements.

- Overlay module processing. The linkage editor prepares modules for overlay by assigning relative locations within the module to overlay segments and by inserting tables to be used by the overlay supervisor during execution.

- Options and error messages. The linkage editor can:

    1. Process special options that override automatic library calls or the effect of minor errors.

    2. Produce a list of linkage editor control statements that were processed.

    3. Produce a module map of control sections in the output load module.

Linkage Editor Control Statements

The linkage editor performs editing functions either automatically or as directed by control statements. These editing functions provide for program modification on a control section basis. That is, they make it possible to modify a control section within an object or load module, without recompiling the entire source program.

The editing functions can modify either an entire control section or external symbols within a control section. Control sections can be deleted, replaced, or arranged in sequence; external symbols can be deleted or changed (external symbols are control section names, entry names, external references, named common areas, or pseudo registers).

Whatever function is used, it is requested in reference to an input module. The resulting output load module reflects the request. That is, no actual change, deletion, or replacement is made to an input module. The requested alterations are used to control linkage editor processing.

Linkage editor control statements are placed in the input sequence along with object decks. The placement of the control statements governs the action of linkage editor processing. The following sections describe the linkage editor control statements and their placement in the input sequence.

NAME Statement

The NAME statment specifies the name of the load module created from the preceding input modules, and serves as a delimiter for input to the load module. As a delimiter, the NAME statement allows multiple load module processing in the same job. If the name specified matches a member name already existing in the load library, the new module will replace the existing load module (see the REPLACE parameter in the section ´Parameters which may be Specified for LKED´ for further information.) The format of the NAME statement is shown below.

```
NAME     {membername}
```

membername
: is the name to be assigned to the load module that is created from the preceding input modules. It is from one to eight alphanumeric characters in length and it must begin with an alphabetic character.

ALIAS Statement

The ALIAS statement specifies additional names for the output library member. Up to 16 names can be specified on one ALIAS statement, or separate ALIAS statements for one library member. The names are entered in the directory of the work file library along with the member name specified on the NAME statement. An ALIAS statement can be placed before, between, or after object modules or other control statements. It must precede the NAME statement used to specify the member name. If the alias is also an entry point name in the load module, calling the load module by alias name will cause execution to begin at that entry point in the load module. If the alias name is not an entry point name, the main entry will be used. The format of the ALIAS statement is shown below.

```
ALIAS    {aliasname}  [,aliasname,...]
```

aliasname
: specifies an alternate name for this load module. It can be from one to eight characters in length and must begin with an alphabetic character.

ENTRY Statement

The ENTRY statement specifies the symbolic name of the first instruction to be executed when the program is called by its module name for execution. If more than one ENTRY statement is encountered, the first statement specifies the main entry point; all other ENTRY statements are ignored. An ENTRY statement can be placed before, between, or after object modules or other control statements. It must precede the NAME statement for the load module. If an ENTRY statement is not present in the input for a load module, the linkage editor will assign the entry point as the first location of the first module (i.e. object deck or INCLUDEd load module) processed in the load module. In an overlay program, the entry point must be in the root segment. The format of the ENTRY statement is shown below.

```
ENTRY    {entryname}
```

entryname
: is the name of either a control section or an entry name in the input stream to the linkage editor.

INCLUDE Statement

The INCLUDE statement specifies load modules that are to be used as additional sources of input to the linkage editor. INCLUDE statements are processed in the order in which they appear in the input. An INCLUDE statement can be placed before, between, or after object modules or other control statements. It must precede the NAME statement for the load module. The format of the INCLUDE statement is shown below.

```
INCLUDE  {name} (membername [,membername,...])
         {unit}
```

name
: specifies the name appearing in the NAME= subparameter of the UNIT= parameter of a /FILE statement which defines a load module work file library (see Job Set-up later in this chapter for more details).

unit
: specifies the unit number of a /FILE statement which defines a load module work file library (see Job Set-up later in this chapter for more details).

membername
: specifies a load module name or list of load module names separated by commas which is to be used as additional input in creating the output load module. Note, that the load module(s) must have been created using REEDIT (see the section on LKED parameters later in this chapter).

CHANGE Statement

The CHANGE statement causes an external symbol to be replaced by the symbol in parentheses following the external symbol. The external symbol to be changed can be a control section name, an entry name, or an external reference. More than one such substitution may be specified on one CHANGE statement. The CHANGE statement must be placed immediately before either the object module containing the external symbol to be changed, or the INCLUDE control statement specifying a load module which is being used as additional input. The format of the CHANGE statement is shown below.

```
CHANGE   externalsymbol(newsymbol) [,externalsymbol(newsymbol),..]
```

externalsymbol
: is the control section name, entry name, or external reference that is to be changed.

newsymbol
: is the name to which the external symbol is to be changed.

REPLACE Statement

The REPLACE statement may be used to replace one control section with another, delete a control section, or delete an entry name. When a control section is replaced, all references within the input module to the old control section are changed to the new control section. Any external references to the old section from other modules from other modules are unresolved unless changed.

When a control section is deleted, the control section name is also deleted from the external symbol dictionary unless references are made to the control section from within the input module. If there are any such references, the control section name is changed to an external reference. External references from other modules to a deleted control section also remain unresolved.

When deleting an entry name, the entry name is changed to an external reference if there are any references to it within the same input module.

The REPLACE statement must immediately precede either (1) the object module containing the control section or entry name to be replaced or deleted, or (2) the INCLUDE statement specifying the load module containing the control section or entry name to replaced or deleted. The END record in the immediately following object module or the end-of-module indication in the load module terminates the action of the REPLACE statement. The format of the REPLACE statement is shown below.

```
REPLACE  ⎧csectname1[(csectname2)]⎫
         ⎨                        ⎬ ,...
         ⎩entryname               ⎭
```

csectname1
: is the name of the control section to be deleted or replaced. If only csecname1 is present, the control section is deleted. If csectname2 is also present, the control section will be replaced.

csectname2
: is the name of the control section replacing csectname1.

entryname
: is the entry name to be deleted.

OVERLAY Statement

The OVERLAY statement indicates either the beginning of an overlay segment, or the beginning of an overlay region (see the section titled Overlaid Modules for further information on overlaying). Since a segment or a region is not named, the programmer identifies it by giving its origin (or load point) a symbolic name. This name is then used on an OVERLAY statement to signify the start of a new segment or region. The OVERLAY statement must precede the object records of the segment, the INCLUDE statements specifying the files containing the object deck(s) of the segment, or the INSERT statements specifying the control sections to be positioned in the segment. No OVERLAY statement should precede the root segment. The format of the OVERLAY statement is shown below.

```
OVERLAY   symbol[(REGION)]
```

symbol
: is the symbolic name assigned to the origin of a segment. This symbol is not related to external symbols in a module.

(REGION)
: is coded as shown and indicates that this is the origin of a new region.

INSERT Statement

The INSERT statement repositions a control section from its position in the input sequence to a segment in an overlay structure (for more information on overlaying see the section titled ´Overlaid Modules´ later in this chapter). However, the sequence of control sections within a segment is not necessarily the order of the INSERT statements. The INSERT statement must be placed in the input sequence following the OVERLAY statement that specifies the origin of the segment in which the control section is to be positioned. If the control section is to be positioned in the root segment, the INSERT statement must be placed before the first OVERLAY statement. The format of the OVERLAY statement is shown below.

```
INSERT   csectname[,csectname,...]
```

csectname
: is the name of the control section to be repositioned. A particular control section can appear only once within a load module.

Parameters which may be Specified for LKED

The parameters which may be specified for a linkage editor run are described below. The parameters are placed, separated by blanks, on one or more records which have the characters "/JOB" starting in column 1. The /JOB records are placed immediately after the /LOAD statement which invokes LKED, and as many may be used as are required to hold all the parameters to be specified. The parameters are:

```
MAP        SEARCH       PRINT       LIST       STATS
NOMAP      NOSEARCH     NOPRINT     NOLIST     NOSTATS

FORMAT      REEDIT       MAXMEM=n         SYSLMOD=n
NOFORMAT    NOREEDIT          174                 3

SYSUT=n      SYSLIB=library             REPLACE
     4              (library list)
                    SYS.FORTLIB
```

MAP | NOMAP
: MAP indicates that LKED is to produce a map of the control sections and entry points, giving their addresses relative to the beginning of the load module. NOMAP suppresses this function. MAP is the default on batch, and NOMAP is the default on the terminal.

SEARCH | NOSEARCH
: SEARCH indicates that one or more subroutine libraries (as specified by the SYSLIB parameter, described below) are to be searched for any unresolved external references after all the supplied object text has been read (weak external references will not be searched for). NOSEARCH suppresses this action. SEARCH is the default.

PRINT | NOPRINT
: If PRINT is specified, LKED will print the length and entry point of the load module after it has been created. NOPRINT, the default, suppresses these messages.

LIST | NOLIST
: LIST causes LKED to print out each linkage editor control statement (NAME, ALIAS, etc.) as it is encountered. NOLIST, which suppresses this printing, is the default.

STATS | NOSTATS
: If STATS is specified, the linkage editor will print out the block numbers of the SYSLMOD file which were used for the load module created. The default, NOSTATS, suppresses these messages. This is useful when the user wishes to see how full a load module library is getting.

FORMAT | NOFORMAT
: If FORMAT is specified, the linkage editor will create a new directory for the SYSLMOD file before writing the load module onto it. Anything previously put in this file will be destroyed. FORMAT must be used when a load module is being placed in a newly created file, and it may also be used to overwrite the contents of an old file. The size of the directory created is determined by the MAXMEM parameter (see below). The default is NOFORMAT. If more than one load module is being created in a single linkage

editor run, and FORMAT has been specified, FORMAT will be assumed for the first load module only - NOFORMAT will be assumed for all subsequent load modules created in that job.

REEDIT | NOREEDIT

REEDIT causes the linkage editor to include in the load module the CESD (composite external symbol dictionary), which contains all the control section names, entry points, and other symbols. If NOREEDIT is specified, the CESD will not be written out, saving a small amount of disk space (usually about one block per load module). However, if NOREEDIT has been specified for a load module, the module cannot be loaded during SYSLIB search by LOADER or LKED, and it cannot be specified on a linkage editor INCLUDE statement. Therefore, if a subroutine library is being created, REEDIT (which is the default anyhow) must be in effect.

MAXMEM

The MAXMEM parameter is used to specify the size of the directory for the SYSLMOD file. The size is expressed as the greatest number of names (load module names plus aliases) which the directory will be required to hold. The default is 174. This parameter will be ignored unless FORMAT has also been specified.

SYSLMOD

The SYSLMOD parameter indicates which i/o unit is to be used for the output load module (see the discussion under Preparing Load Module Libraries, below). The unit must be given as a unit number between 1 and 4, the default being 3.

SYSUT

The SYSUT parameter indicates which i/o unit is to be used as a work area by the linkage editor. The unit must be a DISK file with blocksize 512 and must be given as a unit number between 1 and 4. It is not necessary to supply a /FILE statement for SYSUT or to provide the SYSUT parameter, as the system will provide a large work file on unit 4 when the linkage editor is invoked, unless the user has explicitly provided a /FILE statement for unit 4.

SYSLIB

The SYSLIB parameter specifies the subroutine library or libraries to be searched for unresolved references after the object text has been read in. A single library or a list of libraries enclosed in parentheses and separated by blanks may be specified. Each library may be a unit number, a unit ddname, or the name of a system library. System library names are of the form SYS.libname, where libname is the name of an existing system library, such as FORTLIB or TEKLIB. The libraries will be searched in the same order as they are specified, with the leftmost being searched first. If SYSLIB is not specified, the default of SYSLIB=SYS.FORTLIB will be used. Subroutine libraries to be supplied by the user must be created by the linkage editor.

REPLACE

If REPLACE is specified, the linkage editor will attempt to fit the new load module into the space occupied by an existing load module of the same name. If REPLACE is not specified, or if no load module of the same name exists, the new load module will be placed in the SYSLMOD file starting with the next free block. Since this will eventually use up all the space in the file, it is best to use REPLACE whenever possible. If the new load module does not fit in the old module's space, the linkage edit must be rerun without REPLACE. The default is to put the module in the free space.

Linkage Editor Job Setup

A linkage editor job consists of the following sequence of input records:

```
       ____________
      |
      |   /FILE statements, if any.
      |____________
      |
      |   /LOAD LKED
      |____________
      |
      |   /JOB records with LKED parameters, if any.
      |____________
      |
      |   Object input and linkage editor control
      |   statements (ENTRY, ALIAS, OVERLAY, etc.),
*     |   except for NAME statement.
      |____________
      |
      |   NAME statement.
      |____________
```

For example, the following job will allocate the file UR.XYZ100.LOAD.PROGRAM, format it as a load module library, and place a load module called PROG with the alias name JOKES into it:

```
/FILE UNIT=(3,DISK),DSN=UR.XYZ100.LOAD.PROGRAM,VOL=SYSFIL,
/      DISP=(NEW,KEEP),SPACE=(TRK,6)
/LOAD LKED
/JOB FORMAT MAXMEM=25 STATS LIST MAP
  .
  .   object decks
  .
  ALIAS JOKES
  NAME  PROG
```

The following example, on the other hand, adds three load modules to an existing load module library. A user-supplied subroutine library is provided in addition to the system FORTLIB library. Note that the load module SUBR1 will be given the alias SUB1.

```
/FILE UNIT=(DISK,NAME=SUBRLIB),VOL=SYSFIL,DISP=SHR,
/      DSN=UR.XYZ100.SUBLIB
/FILE UNIT=(3,DISK),VOL=SYSFIL,DISP=OLD,
/      DSN=UR.XYZ100.LMLIB
/LOAD LKED
/JOB STATS MAP LIST
/JOB SYSLIB=(SUBRLIB SYS.FORTLIB)
  ALIAS SUB1
  .
  .   object
  .
  NAME SUBR1
  .
  .   object
  .
  NAME SUBR2
  .
  .   object
  .
  NAME  SUBR3
```

Preparing Load Module Libraries

Work files which contain load modules are divided into two sections. The first section is the file directory which contains an entry for each load module and alias in the file. Each entry contains information specific to that load module - the load module name (or alias name), location in the file, entry point for the load module, etc. The second section of the file contains the actual load modules. The location of each load module is specified in the associated directory entry.

Before a load module can be placed into a new work file, the work file must be formatted. That is, skeleton directory entries must be built into the first section of the file. This can be accomplished using the FORMAT and MAXMEM parameters of the linkage editor when linking the first load module into the file. As described above in the section ´Parameters which may be Specified for LKED´, specifying FORMAT causes the linkage editor to format the directory building the number of entries as specified in the MAXMEM parameter. On subsequent link edits these parameters must not be present since the file directory will be re-formatted, erasing information already present.

Use of REP Cards

The formats of the object deck records accepted by the linkage editor are documented in Appendix A of this manual. All records are 80 byte card images and are thus referred to as "cards".

In addition to the standard object deck cards, the linkage editor accepts one card which is meant to be typed in by the user and inserted into the object input as needed. This card is the REP card which allows the user to replace text loaded from previously encountered TXT cards or to place data into areas of storage assigned to a control section being loaded for which no TXT cards exist. The format of the REP card is exactly the same as that accepted by the loader. The user is referred to page 195 for a complete description of the REP card.

Executing Load Modules

After a load module has been placed in a user load library it may be executed using the /LOAD statement (for a full description of the /LOAD statement the user is referred to the VPS Handbook). The parameters which must be specified when executing a user load module are shown below.

```
/LOAD  ( lmname,FILE=unitnumber )
       (              name      )
```

lmname
specifies the name of the load module (or alias) to be executed.

FILE=
specifies either the unit number of the /FILE statement defining the user load library <u>or</u> the ddname (NAME parameter) specified on the /FILE statement defining the user load library.

For example, assume that the load module CHECKERS has been link edited into the user load library UR.XYZ123.GAMELIB. The following job will execute the load module.

```
.
.                <-- (other /FILE statements as needed by CHECKERS)
.
/file unit=(disk,name=joblib),dsn=ur.xyz123.gamelib,disp=shr,
/ vol=sysfil
/load (checkers,file=joblib)
```

Note that in this example VPS will assign a unit number.

The LMLIST Program

The LMLIST program is an interactive program which can be used to list general or detailed information pertaining to a user load library.

To execute LMLIST the user must set up a job consisting of a /FILE statement defining unit 3 as the load library to be inspected followed by a /INCLUDE statement including the file LMLIST. The file UR.XYZ123.GAMELIB would be inspected if the following job were executed.

```
   /file unit=(3,disk),dsn=ur.zxyl23.gamelib,disp=shr,vol=sysfil
   /inc lmlist
```

When LMLIST is executed the user will be prompted to determine the type of listing desired with the message -

```
   FULL OUTPUT?
```

If NO is the reply LMLIST will list the number of blocks in the file, the blocks used, the blocks free, the number of directory slots, and the directory slots free. If the reply is YES, the user will receive all of the above information as well as detailed information on all of the load modules in the file.

Overlaid Modules

Ordinarily, when a load module produced by the linkage editor is executed, all the control sections (i.e. main program, subprograms, COMMON areas, etc.) of the module remain in virtual storage throughout execution. The length of the load module is, therfore, the sum of the lengths of all of the control sections. When storage space is not at a premium, this is the most efficient way to execute a program. However, if a program approaches the limits of the virtual storage available, the programmer should consider using the overlay facility of the linkage editor.

In most cases, all that is needed to convert an ordinary program to an overlay program is the addition of control statements to structure the module. The programmer chooses the overlayable portions of the program, and the system arranges to load the required portions when needed during execution of the program.

When the linkage editor overlay facility is requested, the load module is structured into segments so that, at execution time, certain control sections are loaded only when referenced. When a reference is made from an executing control section to another, the system determines whether or not the code required is already in virtual storage. If it is not, the code is loaded dynamically and may overlay an unneeded part of the module already in storage.

A segment may be a main program, a single subroutine, or a group of subroutines. There are two types of segments recognized by the linkage editor, ROOT segments and OVERLAY segments. A root segment is a segment which must always remain in virtual storage and cannot be overlaid by another segment. Main programs and blank and named COMMON must be root segments. An overlay segment is a segment that may reside on magnetic disk, be loaded into virtual storage for execution when referenced, and subsequently be overlaid by another overlay segment.

Overlay segments may be used once or many times in one program execution. The only requirement for multiple use of an overlay segment is that the subroutine(s) contained in the segment be serially reusable. This means that the subroutine(s) must not be written in such a manner as to leave results or program switch settings in the subroutine when control returns to the calling program. In order for a routine to be reusable, all results and switches must be passed back to the calling program from the routine via argument lists or COMMON. In effect, a program cannot 'remember' results or switch settings between calls for it to be reusable.

The most successful overlays are achieved in those cases where a relatively small main program calls many large subroutines. The customary procedure for planning overlay structures is to draw a simple tree diagram of the root segment and the various overlay segments. This diagram will aid in estimating virtual storage requirements and will help avoid obvious errors in overlay structures. The root segment is drawn as the 'trunk' of the tree and each overlay segment is drawn as the 'branch' of the tree. Suppose a load module consists of a main program (20,000 bytes) a blank COMMON (30,000 bytes), and three subroutines each called from the main program - SUB1 (35,000 bytes), SUB2 (40,000 bytes), and SUB3 (45,000 bytes). The example below shows a tree diagram for this load module.

```
                                                      *   <-
                                *                     *     |
 Overlay  *            Overlay  *            Overlay  *     |
Segment 1 *           Segment 2 *           Segment 3 *     |
 (SUB1)   *            (SUB2)   *            (SUB3)   *     |
 35,000   *            40,000   *            45,000   *     |
          *                     *                     *     | Longest
          *                     *                     *     | path =
     -->  *********************************************     | 95,000
    |                           *                           | bytes
    |                           *                           |
 Overlay                        * Root segment              |
 Node A                         * (Main + COMMON)           |
                                * 50,000 bytes              |
                                *                           |
                                *                        <-
```

An overlay node is a point at which an overlay segment joins the root segment (or joins another overlay segment in a more complex structure). An overlay node is also called a 'load point' as it defines a location in virtual storage where an overlay segment will be loaded from magnetic disk. The load point is important as it is used to define overlay structures to the linkage editor. The naming of overlay nodes allows the user to have more than one node in an overlay structure, if desired.

Root segments, overlay segments, and load points are defined by the use of linkage editor control statements defined above. The principal control statements are INSERT and OVERLAY. The following is an example of the input sequence needed to produce the overlay structure defined in the tree diagram above.

```
  insert main
  overlay a
  insert sub1
  overlay a
  insert sub2
  overlay a
  insert sub3
  .
  .                    <-- (Object decks for MAIN, SUB1, SUB2, SUB3)
  .
```

In this example, MAIN is identified as the root segment as it is named on an INSERT statement that precedes the first OVERLAY statement. The first OVERLAY statement defines load point A at the end of the root segment. SUB1 will be loaded at load point A since the INSERT statement for SUB1 directly follows the OVERLAY A statement. The second OVERLAY statement informs the linkage editor that whatever follows will also be loaded at load point A. Thus SUB1, SUB2, and SUB3 are all separate segments having the same load point and therefore, cannot be in virtual storage concurrently. Note that the order in which the overlay segments are defined does not imply that they will be executed in the same order.

In the previous example all three subroutines were called from the main program. However, it is common practice for a subroutine to call one or more additional subroutines. Under this type of program organization, an important rule must be followed when overlay structures are being defined. If one subroutine calls another, the overlay structure must be defined in such a manner that both subroutines can be physically present in virtual storage at the same time.

Two segments that can be in virtual storage at the same time are said to be ´inclusive´. In our previous example, the root segment is inclusive with each of the overlay segments as it will be in virtual storage at the same time as each of the subroutines. However, the called subroutines are ´exclusive´ with each other as only one of the subroutines can be in virtual storage concurrently.

Assume that a main program (15,000 bytes) defines a COMMON area (10,000 bytes). The main program calls SUBA (45,000 bytes). SUBA calls SUBB (20,000 bytes). The main program also calls SUBC (30,000 bytes) and SUBC calls SUBD (40,000 bytes) and SUBE (38,000 bytes). Since SUBA calls SUBB both subroutines must be able to reside in virtual storage concurrently. One way to accomplish this is to place both SUBA and SUBB in the same overlay segment. If we assume for the sake of example that the available virtual storage <u>above</u> the root segment is 70,000 bytes, it becomes apparent that no other subroutine can be placed in this overlay segment. Since SUBC calls SUBD, these subroutines could also be placed in a separate overlay and still satisfy our mythical storage requirement. But SUBC also calls SUBE and all three subroutines cannot fit in the same segment. Therefore, each subroutine must be placed in a separate segment.

This structure can be defined in the following manner. SUBC is defined as the second overlay segment in the structure having its origin at node ONE. Next, a second overlay node (or load point), called TWO, is defined at the end of SUBC. Now two additional overlay segments, segment three containing SUBD and segment four containing SUBE can be defined. These segments will be loaded above SUBC (the second overlay segment) and will overlay each other. The following is a tree diagram showing the now complete overlay structure.

```
          *                                              <-
          *                     *                          |
 Overlay  *            Overlay  *            Overlay  *    |
Segment 3 *           Segment 4 *           Segment 1 *    |
 (SUBD)   *            (SUBE)   *            (SUBA,B) *    |
 40,000   *            38,000   *            65,500   *    |
          *                     *                     *    |
     -->  ***********************                     *    |
    |               *                                 *    |  Longest
    |               * Overlay                         *    |  path =
 Overlay            * Segment 2                       *    |  95,000
 Node TWO           * (SUBC)                          *    |  bytes
                    * 30,000                          *    |
                    *                                 *    |
               -->  ***********************************    |
              |                      *                     |
              |                      *                     |
           Overlay                   * Root segment        |
           Node ONE                  * (Main + COMMON)     |
                                     * 25,000 bytes        |
                                     *                     |
                                     *                    <-
```

The linkage editor control statements required to define this structure are:

```
insert main
overlay one
insert suba
insert subb
overlay one
insert subc
overlay two
insert subd
overlay two
insert sube
.
.                      <--- (Object decks for Main + subroutines)
.
```

Chapter 6: LOADER

LOADER is a standard relocating loader which reads in object decks produced by compilers or assemblers, places the object text into user storage, optionally searches one or more subroutine libraries for unresolved references, and transfers control to the resulting loaded program. The program must be one which is designed to run in the native VPS environment (e.g. any Fortran program compiled by Fortran G (FORTG), any Assembler program using the macros described in the VPS System Facilities for Assembler Programmers manual). Programs which require the simulated OS environment (e.g. PL/I Optimizer (PLIOPT) programs, Assembler programs which use OS macros) may not be run using LOADER.

Parameters are passed to LOADER via the PARM parameter on the /LOAD statement (see the VPS Handbook for a complete description of the /LOAD statement). If more than one parameter is to be passed, the parameters must be separated by either blanks or commas. If a parameter string is also to be passed to the loaded program, it must follow the LOADER parameters in the PARM field and must be separated from them by a slash ("/").

Parameters which may be Specified for LOADER

The following parameters may be specified in the PARM field. If more than one parameter is specified, the parameters must be separated by blanks. The default values are underlined.

ALPHLIST | <u>NOALPHLIST</u>

The ALPHLIST parameter causes an alphabetical list of entry points and csect names to be printed when LOADER has finished loading the program. The address corresponding to each symbol is also printed.

CLEAR=hh

The CLEAR parameter allows the user to specify the hexadecimal value to which every byte of available user storage is to be set before loading is begun. The default is CLEAR=00.

<u>GO</u> | NOGO

GO/NOGO indicates whether the program should be executed after it has been loaded. If NOGO is specified, execution will not be attempted, even if no errors occur during loading. In fact, if NOGO is specified, loading will not occur at all unless MAP or ALPHLIST has been specified. If GO is specified (the default), execution will be attempted provided no serious errors have occurred during loading (see LET, below, for the case of unresolved external references).

LET | <u>NOLET</u>

Normally, if a program loads and some external references remain unresolved (other than weak external references), LOADER will not attempt to execute the program. If LET is specified, and no more serious error has occurred, execution will be attempted despite the unresolved references.

MAP | <u>NOMAP</u>

If MAP is specified, LOADER will print a map giving the lengths and locations of all csects loaded, the locations of all entry points, the lengths and locations of all commons allocated, and the values of all pseudo-registers (external dummy sections).

PRINT | NOPRINT

NOPRINT suppresses the informatory messages printed at the end of loading which give the total length and entry point of the loaded program. PRINT is the default on batch, NOPRINT is the default on terminal.

<u>SEARCH</u> | NOSEARCH

NOSEARCH prevents the loader from attempting to find unresolved references in the subroutine library (see SYSLIB, below).

SYMTAB | <u>NOSYMTAB</u>

If SYMTAB is specified, LOADER will build a csect called SYMBLS at the end of the loaded program containing all the csect and entry point names present in the program. Each entry in the table is 12 bytes long and consists of the 8 byte symbol followed by the 4 byte address corresponding to that symbol. The high order byte of the address word is hexadecimal 00 if the symbol is a csect name and 03 if it is an entry point. The last entry of the table is followed by a single byte with hex value 01.

SYSIN=unit

The SYSIN parameter is used to specify the unit to be read by the loader. The unit may be specified as either a unit number or a ddname. The default is SYSIN=5.

SYSIN2=unit

The SYSIN2 unit, if specified, will be read by the loader after end-of-data has been reached on SYSIN. If SYSIN2 is not specified, no SYSIN2 input will be read.

SYSLIB=subroutine library | SYSLIB=(list)

The SYSLIB parameter specifies the subroutine library or libraries to be searched for unresolved references after the object decks have been read in. A single library or a list of libraries enclosed in parentheses and separated by blanks may be specified. Each library may be a unit number, a unit ddname, or the name of a system library. System library names are of the form SYS.libname, where libname is the name of an existing system library, such as FORTLIB or TEKLIB. The libraries will be searched in the same order as they are specified, with the leftmost being searched first. If SYSLIB is not specified, the default of SYSLIB=SYS.FORTLIB will be used. Subroutine libraries supplied by the user must be created by the linkage editor (see LINKEDIT, page 175).

TEST

If TEST is specified, the loader will force an invalid operation interrupt if a serious error occurs during loading, rather than the normal action of printing an error message. This is not likely to be of much use to anyone who does not have a source listing of the loader.

ZSECT | <u>NOZSECT</u>

If a map is being printed (see MAP, above), and ZSECT is specified, LOADER will examine the beginning of each csect for an identifier of the form produced by the VPS macro ZSECT (this is of interest only to assembler programmers). If such an identifier is found, its character string portion will be printed on the same line of the map as the csect name. If ZSECT is set and no such identifier is found for a csect, an ellipsis, "...", will be printed in its place for that csect. The format of this identifier, in Assembler language, is:

```
 NAME   ZSECT
+NAME   CSECT
+       DC     X´76´           INDICATE PRESENCE OF ZSECT
+       DC     AL1(Y-X)        LENGTH OF CHARACTER STRING
+X      DS     0C              START OF CHAR STRING.
+       DC     C´ ´
+       DC     CL8´NAME´       CSECT NAME
+       DC     C´ ´
+       DC     CL5´16.41´      TIME OF ASSEMBLY
+       DC     C´ ´
+       DC     CL7´28SEP77´    DATE OF ASSEMBLY
+       DC     C´ ´
+Y      DS     0C              END OF STRING
```

<u>Job Set-up for LOADER</u>

Unless SYSIN has been redefined (see SYSIN parameter, above), the job set-up which is used with the loader is

```
/LOAD LOADER,PARM=´...´
  .
  .   object decks
  .
/DATA
  .
  .   data (if any) for loaded program
  .
```

The /DATA statement is used to indicate to the loader that the end of the object input has been reached. Reading a /DATA statement on SYSIN or SYSIN2 is considered by LOADER as equivalent to reaching end-of-data on that unit. Thus, when the loaded program goes into execution it can continue reading that same unit, starting with the first record following the /DATA. We give here a couple of examples of jobs using LOADER.

This job loads the object decks saved as library file PROGOBJ, resolves subroutine references from a user subroutine library and the system libraries FORTLIB and TEKLIB, prints a map, and executes the loaded program. The program reads the data in library file PROGDATA.

```
/FILE UNIT=(DISK,NAME=SUBLIB),VOL=SER=SYSFIL,
/     DSN=UR.XYZ100.SUBS,DISP=SHR
/LOAD LOADER,
/     PARM='MAP SYSLIB=(SUBLIB SYS.FORTLIB SYS.TEKLIB)'
/INC PROGOBJ
/DATA
/INC PROGDATA
```

This job loads the stuff in files OBJ1 and OBJ2 and executes the loaded program, which reads the file DATA6 from unit 5. The loader is instructed not to search any libraries for unresolved external references and to execute the loaded program even if such references exist.

```
/FILE UNIT=(15,LIBIN),DSN=OBJ1
/FILE UNIT=(16,LIBIN),DSN=OBJ2
/LOAD LOADER,PARM='SYSIN=15 SYSIN2=16 LET NOSEARCH'
/INC DATA6
```

Current System Libraries

The following system libraries are currently available:

CLCMPLIB - subroutines used to produce output for the Calcomp plotter.

FORTLIB - standard and locally written Fortran subroutines (see the VPS Guide to Programming Languages for descriptions of the latter routines).

PLIXLIB - the PL/I Optimizer resident and transient libraries (probably of little use to anyone using LOADER).

TEKLIB - subroutines used to drive the Tektronix graphics terminals (the Tektronix Terminal Control System (TCS) and Preview packages).

Format of Object Deck Cards Accepted by LOADER

The formats of the standard records accepted by the loader are described in Appendix A of this manual. All records are 80 byte card images and are thus referred to here as "cards".

In addition to the standard object deck cards, the loader accepts four cards which are meant to be typed in by the user and inserted into the object input where needed. These are:

LDT (Loader Terminate) card:

The LDT card terminates loader input immediately and optionally specifies the symbol which is to be the entry point of the loaded program.

col	length	contents
1	1	X´02´
2	3	´LDT´
5	12	blank
17	8	Entry point symbol or blank.
25	56	Ignored.

REP (Replace) card:

The REP card allows the user to replace text loaded from previously encountered TXT cards or to place data into areas of storage assigned to a csect being loaded for which no TXT cards exist. The REP card cannot store data outside the csect whose ESD ID it bears and must be placed somewhere between the ESD card containing the SD or PC entry for that csect and the corresponding END card. If data from a TXT card is being overlayed, the REP card may not precede that TXT card. If the data being stored is to be relocated, the REP card must appear before the corresponding RLD card. REP cards which replace non-relocatable data (the most common case) are usually placed just before the END card.

REP cards are usually used to avoid reassembling a program when only a few bytes need to be changed (non-assembler programmers rarely, if ever, use REP cards). In the interest of users not getting themselves unnecessarily bunged up, it is recommended that REP cards be used sparingly, and that final modifications always be made to the source program. It is not amusing to crack a bug, make the fix via REP cards without making the corresponding changes in the source, and then a few months later replace the object deck with a newly created deck and have the old bug come back to life.

col	length	contents
1	1	X´02´
2	3	´REP´
5	2	blank
7	6	Six digit hex address with leading zeroes.
13	2	blank
15	2	Two digit hex ESD ID of csect being repped (usually "01").
17	...	Data to be stored, in groups of 4 digits separated by commas. The last group must be followed by a blank. The rest of the card may be used for comments.

The following is an example of a REP card which stores the value X´5A9E6003´ at X´2C0´ in the csect having the ESD ID X´01´. (Note that the address on a REP card is relative to zero, not relative to the address at which the csect is being loaded. That is, the address to be used is the address which appears in the assembler listing.)

```
     ⅋REP  0002C0  015A9E,6003  FIX THE BUG
```

where the "⅋" represents the hex 02 which must appear in column 1. This may be entered with the XIN mode of the editor (see page 90) or by defining a HEX character for terminal input editing (see /CTL in the VPS Handbook). If the REP card is really being punched on an actual card with a keypunch, this column must be multipunched with 12-2-9 (the ampersand is the 12 punch).

SPB (Set Page Boundary) card:
The SPB card aligns the next input csect to a 4K page boundary. SPB cards must be placed between object decks, not within them.

col	length	contents
1	1	X´02´
2	3	´SPB´
5	76	Ignored.

SLC (Set Location Counter) card:

The SLC card allows the user to set the loader´s location counter to any value within the storage area available for loading. Both a hex value and the name of a previously defined SD, LD, or LR may be given - the address will be computed as the sum of the hex value and the value of the symbol. If either field is blank, its contribution will be taken as zero. Like the SPB card, the SLC card is meant to appear between, not within, object decks.

col	length	contents
1	1	X´02´
2	3	´SLC´
5	2	blank
7	6	Six digit hex value or blank.
13	4	blank
17	8	Symbol or blank.
25	56	Ignored.

The ESD Utility Program

The program called ESD in the VPS library is primarily used to inspect object decks. ESD will print out the contents of ESD records in the same format as used by the Assembler to print its External Symbol Dictionary listing. END cards are also printed out in a special format, and TXT and RLD cards may be formatted if desired. All other object deck records and all non-object records are printed verbatim preceded by their line numbers relative to the beginning of the input to ESD. ESD may also be asked to copy the records it reads to another unit, if desired, and the output from any of the classes of records (ESD, TXT, RLD, END, other) may be produced or suppressed. These options are controlled by the statements:

=COPY=n

$$\text{=LIST=}\ \begin{bmatrix}\text{ESD}\\\text{NOESD}\end{bmatrix}\ \begin{bmatrix}\text{TXT}\\\text{NOTXT}\end{bmatrix}\ \begin{bmatrix}\text{RLD}\\\text{NORLD}\end{bmatrix}\ \begin{bmatrix}\text{END}\\\text{NOEND}\end{bmatrix}\ \begin{bmatrix}\text{XXX}\\\text{NOXXX}\end{bmatrix}\ \begin{bmatrix}\text{ALL}\\\text{NOALL}\end{bmatrix}$$

which may appear anywhere in the data stream. The option chosen by a statement takes effect with the next record. The =COPY statement indicates that copying to unit n is to begin (=COPY and =LIST statements themselves are not copied). The =LIST statement turns the various classes of listing on and off. The options are (defaults underlined):

$\underline{\text{ESD}}$ | NOESD
    Control the printing of information from ESD records.

TXT | $\underline{\text{NOTXT}}$
    Control the printing of TXT card information.

RLD | $\underline{\text{NORLD}}$
    Control the printing of information from RLD cards.

$\underline{\text{END}}$ | NOEND
    Control the printing of information from END cards.

$\underline{\text{XXX}}$ | NOXXX
    Control the printing of non-object records.

ALL | NOALL
    Turn all the above printing options on or off, respectively.

ESD reads from the input stream (unit 5). Thus, all we need type (at *GO level) to print out the contents of the file OBJFILE, for example, is

```
ESD,OBJFILE
```

Chapter 7: COMPARE - File Comparing

The COMPARE utility program is used to locate record differences between two files. A record is read from each file specified and the two records are compared. If a difference is detected the records are printed and a message is produced indicating the line and column location of the mismatch. This process continues until one or both of the files is exhausted. If the records compared are of different lengths the shorter record is padded with blanks and then compared against the longer record.

The COMPARE utility may be used only at the terminal. The format for invocation of the utility is shown below.

```
COMPARE
```

Specifying Program Parameters

After the COMPARE utility has been invoked, the program will request the name of each of the files to be compared with the following message:

```
ENTER FILE 1 PARAMETERS:
```

At this time the user should enter the first file name optionally followed by one or more blanks and the line number in the file where comparing is to begin. If no number is specified, line number 1 is assumed. This may be followed by the starting column in the line where comparison is to begin and the length of the field to be compared. If these parameters are specified, the comparison will be done only for the field specified. Otherwise, the entire line will be compared. A similar message will then appear requesting the same information for the second file.

Note that by entering a question mark (?) to the file 1 message, the parameter format will be displayed.

When the COMPARE utility has finished processing the files a statistics record will be printed indicating the number of records read from each file and the number of mismatches.

COMPARE Example

In the following example the files COMTST1 and COMTST2 are compared. The comparisons will start with the second record of COMTST1 and the first record of COMTST2 starting at column 3 for a length of 77.

```
compare

ENTER FILE 1 PARAMETERS:comtstl 2 3 77

ENTER FILE 2 PARAMETERS:comtst2 1 3 77

LINE 2 AT COLUMN 14:
11122234567890
-------------|
11122234567891

LINE 4 AT COLUMN 3:
5566678896765432
--|
5576678896765432

EOD ON UNIT 1
8 RECORDS READ FROM FILE 1; 7 RECORDS READ FROM FILE 2; 2 MISMATCHES
*GO
.
.
.
```

Chapter 8: FILELIST - Listing Files

The FILELIST utility program may be used on the VPS batch to produce printer-formatted listings of library files complete with header pages, page titles, and a record count.

Input to the program is a list of file names (not the files themselves). Optionally, the user may suppress the header pages by inserting option cards in the input stream. The format of the option card is as follows:

```
NOHeader
Header
```

where: a dash must appear in column 1 followed by one or more blanks followed by NOHEADER, indicating that the header page is to be suppressed, or HEADER, indicating that the header page is to be printed.

If NOHEADER is specified, only the header (or banner) pages are suppressed, the page titles and record counts are still printed. The specified option will remain in effect until another option card is encountered. Unless otherwise specified, header pages will be produced.

Note that the maximum logical record length of files listed with FILELIST is 133 characters.

FILELIST Example

In the following example the job stream shown would cause FILE1 to be listed with headers, FILE2 and FILE3 to be listed without headers, and FILE4 to be listed with headers.

```
/id acctnum
/inc filelist
file1
- noh
file2
file3
- h
file4
```

The identical job stream sent to the VPS batch using the /BQC command (see the VPS Handbook) follows:

```
/bqc j filelist,'file1','- noh','file2','file3','- h','file4'
```

Note that since FILELIST will search for file names anywhere on a line the /BQC command could also be typed as follows:

```
/bqc j filelist, file1,- noh, file2, file3,- h, file4
```

Chapter 9: OS SORT/MERGE Package

Chapter 3 of this manual describes the /SORT command which can be used on VPS to sort library files in memory. VPS also supports the OS SORT/MERGE package which can be used to sort or merge library files as well as work files and tape files. Unlike the /SORT command, the SORT/MERGE package uses intermediate disk work files for sorting so that sort capacity is virtually unlimited. This package has not been modified to run under VPS and is fully documented in the IBM publication *OS SORT/MERGE Programmer´s Guide* (SC33-4007).

Note that the OS SORT/MERGE package may be run as an independent package or invoked from a PL/I or COBOL program. The VPS Guide to Programming Languages describes how to call SORT/MERGE from PL/I and COBOL. This chapter describes how the SORT/MERGE package may be invoked directly under VPS.

The Library File - SORTMERG

The VPS library file SORTMERG contains all the VPS control statements needed to perform standard sorts. Programmer modifiable options on these control statements have been coded as symbolic parameters (see the VPS Handbook) for easy specification using /SET statements. The symbolic parameters used in the file SORTMERG and their default values are shown below.

Parameter	Description
INFILE = NULLFILE	the input file name
INUNIT = LIBIN	the input file type
INRFM = F	the input file record format
INBLK = 80	the input file BLKSIZE
INLREC = 80	the input LRECL
INVOL =	the input volume serial number (for tape files only)
INLP = NL	the input LABEL type (for tape files only)
INPOS = 1	the input file sequence number (for tape files only)
OUTFILE = *2	the output file name
OUTUNIT = LIBOUT	the output file type
OUTRFM = F	the output file record format

OUTBLK = 80	the output file BLKSIZE
OUTLREC = 80	the output file LRECL
OUTDISP = NEW	the output file disposition
OUTVOL =	the output volume serial number (for tape files only)
OUTLP = NL	the output LABEL parameter (for tape files only)
OUTPOS = 1	the output file sequence number (for tape files only)
PRIM = 10	the allocation for each sort work area
SPTYP = TRK	the type of sort work allocation (PRIM)
NSEGS = 4	the number of virtual memory segments (64K) sort is to use
SORTYP = ´/* ´	this parameter should be used <u>only</u> when requesting a crisscross sort (see the Programmer´s Guide) and should be coded as."SORTYP="

The file should be included as part of a job stream preceded by /SET control statements to set pertinent symbolic parameters and followed by the necessary SORT/MERGE control statements.

<u>Calculating Temporary SORT/MERGE Work Space</u>

In the above parameters the PRIM= parameter is used to specify the number of tracks to be allocated to each of the three sort work datasets (note that if SPTYP=CYL is coded PRIM= then denotes the number of cylinders to be allocated). The default allocation of 10 tracks allows for 3600 80 byte records to be sorted. Sorting efficiency is maximized by correctly calculating the amount of sort work allocation for each particular sort. The following formula should be used to calculate sort work space (note that this formula is for 3350 disk drives - currently in use at the Computing Center - the SORT/MERGE Programmer´s Guide contains a full discussion of sort work calculations):

$$\text{PRIM} = \frac{\text{nrec} * \text{lrecl}}{36000} + 2 \qquad \text{(round up to nearest whole number)}$$

where:

nrec = number of records to be sorted
lrecl = record length

SORT/MERGE Control Statements

SORT/MERGE control statements are fully documented in the SORT/MERGE Programmer´s Guide. These control statements provide the package with information on the fields to be sorted, the sort order, the size of the input file, etc. Control statements should be placed directly after the /INC SORTMERG control statement.

Examples

In the following example the library file RANDOM is sorted in ascending order based on columns 1 through 9 of each record and placed in the new library file SORTED. RANDOM contains 3158 80 byte records.

```
/SET INFILE=RANDOM,OUTFILE=SORTED
/INC SORTMERG
  SORT FIELDS=(1,9,CH,A),FILSZ=3158
```

In the next example a tape file containing approximately 9600 100 byte records is sorted into the existing work file UR.UTL100.SORTED.DATA.

```
/SET INFILE=DATA,INVOL=DATA76,INUNIT=TAPE,INBLK=8000,INLREC=100
/SET INRFM=FB
/SET OUTFILE=´UR.UTL100.SORTED.DATA´,OUTUNIT=DISK,OUTDISP=OLD
/SET PRIM=29
/INC SORTMERG
  SORT FIELDS=(3,13,CH,A,50,2,CH,A),FILSZ=E9600
```

File Merging

SORT/MERGE can also merge sorted files into a single file. The procedure and control statements for merging are fully described in the SORT/MERGE Programmer´s Guide. Unlike sorting, the majority of the VPS control statements needed for merging must be specified by the programmer; therefore, no VPS library file exists containing the general statements.

The programmer must provide a /FILE statement for each file to be merged. Each /FILE statement must have NAME=SORTIN0n (where n is 1 for the first file to be merged, 2 for the second, etc.) in the UNIT= parameter. Due to the nature of VPS, when library files are being merged, the record format (RECFM=) should be set to F and block size (BLKSIZE=) set to the LRECL on the /FILE statement for the first input file. A /FILE statement must also be specified for the merged output file having NAME=SORTOUT coded in the UNIT= parameter. These control statements must be followed by the rest of the necessary VPS control statements needed to invoke the package. The job stream is defined as follows:

```
/FILE UNIT=(utype,NAME=SORTIN01),...
/FILE UNIT=(utype,NAME=SORTIN02),...
.
.
.
/FILE UNIT=(utype,NAME=SORTOUT),...
/FILE UNIT=(DISK,NAME=SORTLIB),DSN=UR.CCMM.SORTLIB,DISP=SHR
/FILE UNIT=(TERMOUT,NAME=SYSOUT)
/LOAD OSLOAD,PARM='MEMBER=SORT',NSEGS=4
/JOB JOBLIB=(SORTLIB)
/PARM LIST
  merge control statements
```

Merge Example

In the following example three library files are merged into a new work file. Each library file has a record length of 80 bytes. The output work file is blocked at 118 records per block, an optimum blocking factor for 3350 disk drives.

```
/FILE UNIT=(LIBIN,NAME=SORTIN01),DSN=BUMPERS,DISP=SHR,RECFM=F,
/ BLKSIZE=80
/FILE UNIT=(LIBIN,NAME=SORTIN02),DSN=JACKS,DISP=SHR
/FILE UNIT=(LIBIN,NAME=SORTIN03),DSN=SKATES,DISP=SHR
/FILE UNIT=(DISK,NAME=SORTOUT),DSN=UR.UTL100.SMALL.PERSON.ITEMS,
/ DISP=(NEW,CATLG),SPACE=(TRK,(5)),RECFM=FB,LRECL=80,BLKSIZE=9440
/FILE UNIT=(DISK,NAME=SORTLIB),DSN=UR.CCMM.SM01,DISP=SHR
/FILE UNIT=(TERMOUT,NAME=SYSOUT)
/LOAD OSLOAD,PARM='MEMBER=SORT',NSEGS=4
/JOB JOBLIB=(SORTLIB)
/PARM LIST
  MERGE FIELDS=(1,6,A,20,10,D),FORMAT=CH,FILSZ=E8000
```

Chapter 10: TRCASE - Upper/Lower Case File Conversion

The TRCASE utility program is used to change the case of alphabetic characters within a VPS library file. TRCASE may be used only at the terminal and is invoked using the format shown below.

```
TRCASE
```

Specifying Program Parameters

After the TRCASE utility program has been invoked, the program will request the name of the file to be converted and the type of translation desired with the following message:

```
ENTER FILE NAME AND TRANSLATION TYPE:
```

At this time the user should enter the file name optionally followed by one or more blanks and one of the following options:

Flip
Lower
Upper

where: FLIP is the default and indicates that all upper case characters are to be translated to lower case and all lower case characters are to be translated to upper case. LOWER indicates that all alphabetic characters should be translated to lower case. UPPER indicates that all characters should be translated to upper case. Note that FLIP, LOWER, and UPPER may be abbreviated.

After TRCASE processing has completed, the *2 file (unit 10) will contain the translated version of the file. The *2 file may be saved or replace an existing file using the /SAVE or /RSAVE commands (see the VPS Handbook).

Upper and Lower Case

At this point a short discussion to clarify the terms "upper case" and "lower case" is in order. Under normal operation (full text mode - see the /TEXT command in the VPS Handbook) input entered from a terminal in lower case is translated to upper case EBCDIC before processing and input entered in upper case is translated to lower case. Conversely on output, upper case output is translated to lower case before it is sent to the terminal and lower case output is translated to upper case. Therefore, if a user creates a file using /EDIT (see Chapter 1) all lower case characters entered exist in the file in upper case. Of course, characters entered in upper case exist in the file in lower case. TRCASE translation operates only on the file specified and is independent of terminal translation. Consequently, if a file is translated with TRCASE and the option UPPER is specified, the translated file will be in all upper case characters even though listing the file at a terminal in full text mode will show all lower case characters (FLIPped files will appear reversed and LOWER files will appear in upper case). Note that the FLIP option is useful for script files received from other institutions or to be sent to another computing facility.

TRCASE Example

In the following example the file NODICE is translated to all upper case and is replaced by the translated version.

```
trcase
ENTER FILE NAME AND TRANSLATION TYPE:nodice u
*GO
/rsav nodice
.
.
.
```

Appendix A: Object Deck Card Formats

The formats of the records accepted by the loader and the linkage editor are shown below. All records are 80 byte card images and are thus referred to here as "cards".

ESD (external Symbol Dictionary) card:

col	length	contents
1	1	X´02´
2	3	´ESD´
5	6	blank
11	2	Number of bytes of ESD data (= number of entries X 16).
13	2	blank
15	2	ESD ID of first non-LD item, blank if no such item.
17	16-48	1, 2, or 3 items.
73	8	Sequence field.

Each ESD item:

col	length	contents
1	8	Symbol (blank if PC or blank CM).
9	1	Type: 00 SD control section definition 01 LD label definition (eg, ENTRY) 02 ER external reference 04 PC private code (unnamed csect) 05 CM common 06 PR pseudo-register (external dummy section) 0A WX weak external reference
10	3	Address if SD, PC, LD, or LR, blank otherwise.
13	1	Alignment factor if PR: 00 byte 01 halfword 03 fullword 07 doubleword blank otherwise.
14	3	Length if SD, PC, CM, or PR (zero if SD or PC and length on END card), ID of SD containing name if LD, blank otherwise.

TXT (Text) card:

col	length	contents
1	1	X´02´
2	3	´TXT´
5	1	blank
6	3	Address of first text byte.
9	2	blank
11	2	Number of text bytes.
13	2	blank
15	2	ESD ID for text.
17	1-56	Text data.
73	8	Sequence field.

RLD (Relocation Dictionary) card:

col	length	contents
1	1	X´02´
2	3	´RLD´
5	6	blank
11	2	Number of bytes of RLD data.
13	4	blank
17	8-56	RLD items.
73	8	Sequence field.

RLD item:

col	length	contents
1	2	Relocation pointer (ESD ID of symbol referenced).
3	2	Position pointer (ESD ID of control section containing address constant).
5(1)	1	Flags:

```
0 0 0 0 . . . .  Non-branch (A-con).
0 0 0 1 . . . .  Branch (V-con).
0 0 1 0 . . . .  PR displacement value (Q-con).
0 0 1 1 . . . .  PR cumulative length (CXD).
. . . . 0 1 . .  2 bytes to relocate.
. . . . 1 0 . .  3 bytes.
. . . . 1 1 . .  4 bytes.
. . . . . . 0 .  Positive relocation.
. . . . . . 1 .  Negative relocation.
. . . . . . . 0  Relocation and position pointers
                 present in next item.
. . . . . . . 1  Relocation and position pointers
                 absent in next item - use
                 same pointers as this item.
```

col	length	contents
6(2)	3	Address of address constant being relocated.

END card, format 1:

col	length	contents
1	1	X'02'
2	3	'END'
5	1	blank
6	3	Entry point address or blank.
9	6	blank
15	2	ESD ID of csect containing entry point.
17	12	blank
29	4	Length of SD or PC which had zero length on ESD card.
33	40	Ignored - usually identifies compiler or assembler which created deck and date and time of creation.
73	8	Sequence field.

END card, format 2:

col	length	contents
1	1	X'02'
2	3	'END'
5	12	blank
17	8	Symbolic entry point name.
25	4	blank
29	4	Length of SD or PC which had zero length on ESD card.
33	40	Ignored - usually identifies compiler or assembler which created deck and date and time of creation.
73	8	Sequence field.