Dennis D. Simpson

decsystem10

UTILITIES MANUAL

digital

decsystem10

UTILITIES MANUAL

DEC-10-UTILA-A-D

digital equipment corporation · maynard. massachusetts

1st Edition March 1975

The HOW TO OBTAIN SOFTWARE INFORMATION page, located at the back of this document, explains the various services available to DIGITAL software users.

The postage prepaid READER'S COMMENT form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

CONTENTS

FOREWORD

This manual contains the documentation for the following software:

CREF	Version 51
DDT	Version 35
FILCOM	Version 20A
FILEX	Version 16
GLOB	Version 5A
OPSER	Version 5A
PIP	Version 33A
RUNOFF	Version 10

These utilities are used

1. To obtain cross-referenced listings for all operand-type symbols, user-defined symbols, and/or op codes and pseudo-op codes.

2. To check out and test on-line programs.

3. To compare two versions of a file and output any differences.

4. To convert various core image formats while transferring files.

5. To obtain an alphabetical cross-referenced listing of all global symbols encountered.

6. To facilitate multiple job control for operators.

7. To transfer files from one peripheral device to another.

8. To format documents easily and efficiently.

The change bars in the margins of these documents indicate a change in the software since the last version.

decsystem10

CREF

Cross-Referenced Listing

digital equipment corporation · maynard. massachusetts

The HOW TO OBTAIN SOFTWARE INFORMATION page, located at the back of this document, explains the various services available to DIGITAL software users.

The postage prepaid READER'S COMMENT form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

CROSS-REFERENCE LISTING (CREF)

CREF produces a sequence-numbered assembly listing followed by one to three tables, one showing cross references for all operand-type symbols (labels, assignments, etc.), another showing cross references for all user-defined operators (macro calls, OPDEFs etc.), and another (if the proper switch is specified) showing the cross references for all op codes and pseudo-op codes (MOVE, XALL, etc.). A number sign (#) appears on the definition line of all symbols. The input to CREF is a modified assembly listing file created during a MACRO-10 assembly or FORTRAN IV compilation when the /C switch is specified in the command string.

CREF provides an invaluable aid for program debugging and modification.

1.0 REQUIREMENTS

Minimum Core:	2K pure, 1K impure
Additional Core:	Takes advantage of any additional core available, as necessary.
Equipment:	One input device (normally disk) which contains the modified assembly listing file; one output device (normally the line printer) for the listing.

2.0 INITIALIZATION

.R CREF)	Loads the Cross-Reference Listing program into core.
*	The program is ready to receive a command.

NOTE

If CREF cannot initialize the terminal, it exits.

3.0 COMMANDS

3.1 Command Formats

a. output-dev:filename.ext[ppn]=input-dev:file1.ext[ppn],file2.ext,...

b. progname!

output-dev:filename.ext

The device on which the assembly listing and cross-reference tables are to be printed. If no output file name is specified, the default file name is the same as that specified for the first input file, but with the extension .LST. In such a case, the default device is LPT. However, if an output file name is specified, the default output device is DSK.

input-dev:filename.ext

The device on which the modified assembly listing was written during MACRO-10 assembly. DSK: is assumed if the device is not specified. When looking for the input file, CREF tries the following default extensions in the order listed: .CRF, .LST, .TMP, or a null extension. A missing input file name is given the name CREF. If the input file extension is .CRF or .LST and the /P switch is not included in the command string, the input file is deleted after the output file is successfully closed.

Multiple input files can be specified to be combined into a single CREF listing by separating the input files with commas. Switches affecting the entire listing (/K, /M, /O, and /S) must be specified before the terminator for the first input file. Switches affecting the positioning of an input file are specified with each file.

The ?CRFCFF CANNOT FIND FILE. . . message will be printed for each occurrence of a missing file. If the missing file is not part of a COMPIL-class command file (that is, if it was typed in directly), the command will be aborted, allowing the user to retype the command string. However, if the missing file is part of a COMPIL-class command file, processing will continue for the rest of the existing files in the command string. (Refer to section 4.0 of this document.) Note that if any file is in fact missing, no input file will be deleted.

[ppn]

The disk area on which the files are to be placed (output) or the disk area on which the source files reside (input). If omitted, the default is the user's disk area.

The output device and the input device are separated by an equal sign. If the equal sign is omitted, output defaults occur as described above. Any files specified by the user are for input.

progname!

The user can request CREF to run a system program by typing the program name followed by an exclamation point.

Examples of Commands:

```
.R MACRO↵
```

Load the MACRO-10 Assembler into core.

```
*PTP:,/C=DTA1:TXCALC↵
THERE ARE NO ERRORS

PROGRAM BREAK IS 003771
```

Assemble the program TXCALC from DTA1; writes the object program coding on the paper tape punch; writes a modified assembly listing on DSK: (assumed) and assigns it the filename CREF.LST.

```
[CRFXKC 7K CORE]
*↑C
.R CREF↵
```

Return to the monitor.

Load CREF into core.

```
*↵
```

Select the default assumptions of:

output-dev:	LPT:
input-dev:	DSK:
input filename.ext	CREF.CRF (.LST, .TMP)
output filename.ext	CREF.LST

Equivalent to the command string

LPT:CREF.LST=DSK:CREF.CRF

```
*↑C
.
```

Return to the monitor.

```
.R CREF
*OUTFIL=FIL1,FIL2,FIL3
*LINK!
*FIL1,FIL2,FIL3/G
LINK: LOADING
EXIT
.
```

Make single merged cross-reference file for three program files.

Run LINK10.

3.2 Switches

Switches are used to specify such options as magnetic tape control and list selection. All switches are preceded by a slash (/).

Examples of Switches:

```
.R CREF
```

Load CREF into core.

```
*/M=MTA1:/W
```

Rewind MTA1 and process the first file, listing only the cross references for operand-type symbols (labels, assignments, etc.).

```
*DTA5:SAVE1/Z=
[CRFXKC 7K CORE]
```

Process the file named CREF.LST in the user's area of disk; write the program listing and operand-type cross reference on DTA5 and call the file SAVE1.

```
*↑C
.
```

Return to monitor.

CREF Switch Options

SWITCH	MEANING
A	Advance magnetic tape reel by one file. /A may be repeated.
B	Backspace magnetic tape reel by one file. /B may be repeated.
H	Print help for running CREF.
K	Kill listing of references to basic symbols (labels, assignments, etc.).
M	Suppress listing of references to user-defined operators (Macro calls, OPDEFs, etc.).
O	Allow listing of references to machine and pseudo-operation codes (MOVE, XALL, etc.).
P	Preserve an input file with the extension .CRF or .LST, which is normally deleted.
R	Request the line number at which the listing is to Restart. CREF prints: RESTART LISTING AT LINE: at which time the user types the line number followed by a carriage return. (Such action might be necessary if the line printer ran out of paper, or jammed, etc.)
S	Suppress program listing (list only the selected tables).
T	Skip to logical end of magnetic tape.
W	ReWind magnetic tape.
Z	Zero the DECtape directory (DECtape must be output only).

4.0 MONITOR COMMANDS

CREF-format listing files generated by COMPILE, LOAD, EXECUTE, and DEBUG commands (using the /CREF switch) can be printed on the line printer by typing

.CREF ↵

The CREF command will print out all listing files that are specified in the COMPIL-class command file, nnnCRE.TMP (where nnn is the user's job number). It will also transfer control to a system program if its name is present in the form "progname!". After completion of this operation, nnnCRE.TMP is deleted to prevent the listing files from being listed again by the next CREF command.

The CREF files may also be listed by an R CREF command and a response of "filename" to each asterisk (*) typed by CREF. It is important to note that, if the user uses the R CREF command to list files created by the monitor's COMPIL-class commands, the names of the files to be listed must be typed in response to the asterisks.

5.0 DIAGNOSTIC MESSAGES

CREF Diagnostic Messages

MESSAGE	MEANING
?CRFBTB INPUT BUFFERS TOO BIG	The monitor set up input buffers longer than 203_8. This is not a user error and hopefully will never occur.
?CRFCEF CANNOT ENTER FILE, n dev: file.ext	DTA or DSK directory is full; file cannot be entered; n indicates the cause of the failure and is obtained from the ENTER directory block.
?CRFCFE COMMAND FILE INPUT ERROR, n dev;file.ext	Disk data error while reading nnnCRE.TMP; n is a six-digit (or less) octal number representing the file status word returned from the GETSTS UUO.
?CRFCFF CANNOT FIND FILE, n dev: file.ext	The file cannot be found on the device specified; n indicates the cause of the failure and is obtained from the LOOKUP directory block.

CREF Diagnostic Messages (Cont.)

MESSAGE	MEANING
?CRFCME COMMAND ERROR--TYPE/H FOR HELP	Error in last command string entered. 1. Device name, filename, or extension consisted of non-alphanumeric characters. 2. The project-programmer number was not in standard format (i.e., it was not octal numbers in the form ppn). 3. An undefined switch was specified, switches in parentheses were not separated by commas, or the closing parenthesis was missing. 4. The /Z switch was used on the input side of the command string.
?CRFDNA DEVICE NOT AVAILABLE	Device is assigned to another job.
%CRFIDC IMPROPER INPUT DATA, CONTINUING	Input data is not in CREF format. Output listing continues.
?CRFIMA INSUFFICIENT MEMORY AVAILABLE	Additional core is required for execution but none is available from the monitor.
?CRFINE INPUT ERROR,n dev:file.ext	READ error has occurred on the device.
?CRFOUE OUTPUT ERROR, n dev:file.ext	WRITE error has occurred on the device.
[CRFXKC nK CORE]	Size of low segment in K of core.

Dennis D. Simpson

decsystem10

DDT

Dynamic Debugging Technique

Order No. DEC-10-UDDTA-A-D

digital equipment corporation · maynard. massachusetts

1st Printing, January 1968
2nd Printing (Rev), April 1969
3rd Printing (Rev), June 1969
4th Printing (Rev), November 1969
5th Printing (Rev), August 1970
6th Printing, February 1975

The HOW TO OBTAIN SOFTWARE INFORMATION page, located at the back of this document, explains the various services available to DIGITAL software users.

The postage prepaid READER'S COMMENT form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

CONTENTS

CONTENTS (Cont)

CONTENTS (Cont)

FIGURES

TABLES

CHAPTER 1

INTRODUCTION

DDT-10 (for Dynamic Debugging Technique)* is used for on-line checkout and testing of MACRO-10 and FORTRAN programs and on-line program composition in all DECsystem-10 software systems.

After the user's source program has been assembled or compiled, the user's binary object program (with its symbol table) may be loaded along with DDT. DDT occupies about 2K of core.

By typing commands to DDT, the user may set breakpoints where DDT will suspend execution of his program and await further commands. This allows the user to check out his program section by section. Either before starting execution or during breakpoint stops, the user may examine and modify the contents of any location. Insertions and deletions may be done in symbolic source language or in various numeric and test modes at the user's option. DDT also performs searches, gives conditional dumps, and calls user-coded debugging subroutines at breakpoints.

Symbolic on-line debugging with DDT provides a means for rapid checkout of new programs. If a bug is detected, the programmer makes changes quickly and easily and may then immediately execute the corrected section of his program.

1.1 LOADING PROCEDURE

The user loads the program to be debugged with DDT using the Linking Loader. (The /DEBUG:DDT switch commands LINK-10 to load DDT.) To transfer control to DDT, the user types the monitor command,

```
DDT
```

After DDT responds by skipping two lines, the user may begin typing commands to DDT.

* Historical footnote: DDT was developed at MIT for the PDP-1 computer in 1961. At that time DDT stood for "DEC Debugging Tape." Since then, the idea of an on-line debugging program has propagated throughout the computer industry. DDT programs are now available for all DEC computers. Since media other than tape are now frequently used, the more descriptive name "Dynamic Debugging Technique" has been adopted, retaining the DDT acronym. Confusion between DDT-10 and another well known pesticide, dichloro-diphenyl-trichloroethane (C(14)H(9)CL(5)), should be minimal since they attack different, and apparently mutually exclusive, classes of bugs.

1.2 LEARNING TO USE DDT

This manual is designed to make DDT easy to use. A survey was made of several programmers who use DDT frequently, and it was learned that most debugging is done with a limited set of commands. These basic commands are described in the next chapter. When learning DDT, it is recommended that the reader concentrate on learning to use the commands in Chapter 2. If more detailed information is required, skip ahead to later chapters.

After reading Chapter 2, practice debugging, using the basic commands. This may be all that will ever be needed. Read the following chapters which describe the entire command set in detail; this should be read when the basic commands are understood.

After learning the system, the Summary of Commands, listed by function in Appendix A, will be useful for quickly finding any DDT command.

CHAPTER 2

BASIC DDT COMMANDS

The DDT commands most frequently used by programmers are described in this chapter. Many programs are debugged successfully using only these basic commands.

This chapter introduces the main features of DDT to the uninitiated user. Later chapters describe in detail these basic commands, less frequently used commands and other more complex options.

2.1 EXAMINING STORAGE WORDS

By using DDT, a programmer may examine the contents of any storage word by typing the address of the desired word followed immediately by a slash (/). For example, to type out the contents of a location whose symbolic address is CAT, the user typed,

```
CAT/
```

DDT now types out the contents (preceded and followed by tabs) on the same line[1].

```
CAT/        MOVEM AC,DOG+21
```

The word labeled CAT is now considered to be opened, and DDT has set its location pointer to point to this address.

2.2 TYPE-OUT MODES

The preceding example showed DDT typing out the contents of location CAT as a symbolic instruction with its address file also relative to a symbol. This is the type-out mode in which DDT is initialized. It is also initialized to type all numbers in the octal radix. The user may ask DDT to retype the preceding quantity as a number in the current radix by typing an equal sign (=). For example,[2]

```
CAT/        MOVEM AC,DOG+21  =202040,,6736
```

[1] In this manual information typed out by DDT is underlined to distinguish DDT output from user-typed input.

[2] The two commas indicate the 202400 is in the left half of CAT, and 6736 is in the right half.

DDT has numerous commands which reset the type-out mode permanently, temporarily, or for only one typeout. The modes that can be selected include numeric constants, floating point numbers, ASCII and SIXBIT text modes, and half-word format. Absolute or relative addressing and different radixes may similarly be selected. For example, to change the current type-out mode to ASCII text, the user types the command[3]

```
$T
```

or, to change the current type-out mode to half-word format, he types

```
$H
```

or, to select decimal numbers in his typeouts, he types

```
$10R
```

Using these commands (and others described in Chapter 3), a programmer may examine any location in the mode most appropriate to the information stored there. A semicolon (;) commands DDT to retype the preceding quantity in the current mode. Combining this command with a mode change gives results such as the following:

```
      CAT/      MOVEM AC,DOG+21   $10R;MOVEM AC,DOG+17
or    CAT/      MOVEM AC,DOG+21   $H;202040,,DOG+21
or    TEXT/     ANDM 1,342212(10)  ST;ABCDE
```

2.3 MODIFYING STORAGE WORDS

Once a word has been opened, its contents may be changed by typing the desired new contents immediately following the typeout produced by DDT. A carriage return will command DDT to make the indicated modification and close the word. For example:

```
CAT/      MOVEM AC,DOG+21      MOVNM AC2,DOG+21↵
```

The carriage return simply closes the previously examined register without opening another[1]. The line feed (↓) may also be used to close a word after examining (and optionally modifying it. The line feed commands DDT to

1. echo a carriage return,
2. close the current word (making a modification if one was typed),

---

[3]The terminal keys ALTMODE (ALT), PREFIX (PREFIX), or ESCAPE (ESC) are all equivalent and echo as $.

[1]The carriage return command has the additional property of causing temporary type-out modes to revert to permanent mode.

3. add one to DDT's location pointer, and

4. type out the new pointer value and the contents of that address.

Thus, if a line feed had been used in the previous example, the result would be:

```
CAT/        MOVEM AC,DOG+21        MOVNM AC2,DOG+21↓

CAT+1/        AOBJN XR6,LOOP5
```

Location CAT+1 is now open and may be modified if desired.

The vertical arrow (↑) is similar to the line feed command except that the location counter is decremented by one. Therefore, if the user continued the previous example by typing ↑ the result would be

```
CAT+1/        AOBJN XR6,LOOP5↑

CAT/        MOVNM AC2,DOG+21
```

Location CAT is thus displayed and shows the result of the modification made in the previous example.

The tab (→|) and backslash (\) both close the current register and open the address last typed (whether typed by DDT or the user). However, tab sets DDT's location pointer (.) to this new address while backslash leaves it unaltered. A more complex example may clarify the usefulness of these commands.

```
CAT+1/    AOBJN XR6,LOOP5→|

LOOP5/    CAMGE AC2,TABL(XR6) CAMG AC2,TABL+1(XR6)\ SETZI Ø=401ØØØ,,Ø↓

LOOP5+1/  JUMPL AC3,FAULT JUMPL AC2,FAULT→|

FAULT/    JSRT 4,FAULT
```

2.4 TYPE-IN MODES

The examples in the preceding section showed modifications made as symbolic instructions in a form identical to MACRO-10 machine language. It is also possible to enter various numbers and forms of text.

Octal values may be typed in as octal integers with no decimal point. To be interpreted as a decimal number, an integer must be followed by a decimal point. Numeric strings with numbers following the decimal point imply decimal floating-point numbers. The E-notation may also be used on floating-point numbers. Some examples are:

Octal:	1234	777777777777	-6	0
Decimal integers:	6789.	99999999.	-25.	0.
Floating-point numbers:	78.1	0.249876E-10	-4.00E+20	0.0
Incorrect formats:	76E+2	76.E+2 (instead write 76.0E+2)		

To enter ASCII text (up to five characters, left justified in a word), type a double quote (") followed by any printing character to serve as a delimiter, then type the one to five ASCII characters and repeat the delimiter. For example:

"/ABCDE/	(/ is the delimiter)
"ABCDA	(A is the delimiter)

Note that the mode of a quantity typed in is determined by the user's input format and is unaffected by any type-out mode settings.

2.5 SYMBOLS

The user's symbol tables are loaded by the Linking Loader when it loads programs and DDT. However, initially DDT is set to treat only global symbols (created by INTERNAL and ENTRY pseudo-ops in MACRO-10) as being defined. This means that only global symbols will be used for relative address typeouts and, likewise, only these globals can be referenced when typing in symbolic modifications. In order to make the local symbols within a particular program available to DDT, the user types the program name (this comes from the MACRO-10 TITLE statement or the FORTRAN IV SUBROUTINE or FUNCTION statement) followed by ALTMODE and a colon ($:). For example, the command

```
ARCTAN$:
```

will unlock the local symbols in the program named ARCTAN. This provision in DDT permits the user to debug several related subroutines simultaneously and reference the local symbol table of each independently without fear of multiply-defined local symbols. If the user's program is not titled, the command MAIN.$: will unlock the local symbol table.

NOTE

DDT is not quite so stringent on the use of local symbols as indicated above (see Section 5.6). However, the user is advised to unlock symbols with $: until he is fairly familiar with DDT.

The user may also insert symbols into the symbol table. To insert a symbol with a particular value, type the value, followed by a left angle bracket (<), the symbol, and a colon (:). Some examples are

```
707<CONS:        27<S:        12.1E+<NUMB:        ADR+12<ADRX:
```

To assign a symbol with a value equal to DDT's location pointer, simply type the symbol followed by a colon. For example,

```
XFER+4/      JRST @ TABL(3) BRNCH:
```

will cause BRNCH to be defined with the value XFER+4.

2.6 EXPRESSIONS

DDT permits the user to combine symbols and numeric quantities into expressions by using the following characters to indicate arithmetic operators:

- `+` The plus sign indicates 2's complement addition
- `-` The minus sign indicates 2's complement subtraction
- `*` The asterisk indicates integer multiplication
- `'` The single quote or apostrophe indicates integer division (remainder discarded) – slash cannot be used to indicate division since it has another use in DDT.

As usual in arithmetic expressions, the evaluation proceeds from left to right with multiplication and division performed before addition and subtraction.

2.7 BREAKPOINTS

The breakpoint facility in DDT provides a means of suspending program operation at any desired point to examine partial results and thus debug a program section by section. The simpler facts about breakpoints are presented next; the use and control of conditional breakpoints is deferred to Paragraph 4.2.

2.7.1 Setting Breakpoints

The programmer can automatically stop his program at strategic points by setting as many as eight breakpoints. Breakpoints may be set before the debugging run is started, or during another breakpoint stop. To set a breakpoint, the programmer types the symbolic or absolute address of the word at the location point in which he wants the program to stop, followed by $B. For example, to stop when location 6004 is reached, he types,

```
6004$B
```

Breakpoint numbers are normally assigned by DDT in sequence from 1 to 8. The user may instead assign breakpoint numbers himself when he sets a breakpoint by typing,

```
$nB
```

when n is the breakpoint number (1<n<8). Here are three examples:

```
CAT+3$4B        DOG+1$7B        6004$8B
```

When the programmer sets up a breakpoint he may request that the contents of a specified word be typed out when the breakpoint is reached. To do this, the address of the word to be examined is inserted, followed by two commas, before the breakpoint address. Some examples are

```
DOG,,CAT$3B  AC1,,LOOP+2$B  X,,6004$8B
```

2.7.2 Breakpoint Restrictions

The locations where breakpoints are set may not

1. be modified by the program
2. be used as data or literals
3. be used as part of an indirect addressing chain
4. contain the user mode monitor command INIT
5. be accumulator 0.

2.7.3 Breakpoint Type-Outs

When the breakpoint location is reached. DDT suspends program execution without executing the instruction at the breakpoint location. DDT then types the breakpoint number and the Program Counter value at the time the breakpoint is reached (this value will differ from the typed-in breakpoint address if the breakpoint is executed by an XCT instruction elsewhere in the program). The format of this typeout is as shown in the following examples:

```
$4B>>CAT+3  $7B>>DOG+1  $8B>>6004
```

If the user requested that a specified address be examined at that breakpoint, it will be opened; for example:

```
$3B>>CAT  DOG/        SOJGE 3,GOAT+6
```

2.7.4 Removing and Reassigning Breakpoints

The user may remove a breakpoint by typing,

```
0$NB
```

where n is the number of the breakpoint to be removed. For example:

```
0$2B
```

removes the second breakpoint. All assigned breakpoints are removed by typing

```
$B
```

The user may reassign a breakpoint without formally removing it. Thus, if he has set breakpoint No. 2 at location ADR (via the command ADR$2B) he may reassign No. 2 to LOC+6 by typing LOC+6$2B.

2.7.5 Proceeding From a Breakpoint

Program execution may be resumed (in sequence) following a breakpoint stop by typing the proceed command, $P.

If the user does not wish to stop until the nth time that this breakpoint is encountered he types,

```
N$P
```

Then this breakpoint will be passed n-1 times before a break occurs.

To execute the next instruction, the user types

```
$X
```

which, without an argument, executes the instruction about to be executed after the last breakpoint of the last $X. After the instruction is executed, the PC is updated. (The breakpoint, however, is not moved.) After any number of $X's, $P will always proceed from where the single stepping left off. After executing the instruction, DDT prints out the contents of referenced locations.

2.8 STARTING THE PROGRAM

The program is started by typing

```
$G
```

This starts the program at previously specified starting address .JBSA. (Typically this is the address from the MACRO-10 END statement.) The programmer may start at any other location by typing that address followed by $G. For example:

```
4000$G
```

starts the program at the instruction stored at location 4000. BEGIN$G starts the program at the symbolic location BEGIN.

The start command may also be used to restart from a breakpoint stop when it is not desired to continue in sequence from the point where program execution was suspended.

2.9 DELETING TYPING ERRORS

Any partially typed command may be deleted by pressing the RUB OUT key. This causes DDT to ignore any preceding (unexecuted) partial command. DDT types XXX. The correct command may then be retyped.

2.10 ERROR MESSAGES

If the user types an undefined symbol which cannot be interpreted by DDT, everything typed by the user since DDT's last typeout is ignored, and U is typed. If an illegal DDT command is typed, or a location outside the user's assigned memory area is referenced, the effect is the same except that DDT types a ? instead of a U.

2.11 SUMMARY

As was said in the beginning, these basic commands are sufficient for debugging many programs. Complete descriptions of all DDT commands are explained in the following chapters.

CHAPTER 3

DDT COMMANDS

When DDT is initialized, it is set to type out in the symbolic instruction format with relative addresses, and to type out numbers in octal radix.

3.1 EXAMINING THE CONTENTS OF A PROGRAM STORAGE WORD

To type out the contents of a storage word, the programmer types the address, followed immediately by a slash (/). For example, to examine the contents of a word whose symbolic address is ADR, the user types,

```
ADR/
```

DDT types out the contents on the same line. In this manual, information typed out by DDT is underlined.

```
ADR/        MOVE A,CC1
```

The word labeled ADR is now considered to be opened, and DDT continues to point to this address. The point, or period, character (.) represents DDT's location pointer, and may be used to type out its contents, as in the following command.

```
./          MOVE A,CC1
```

Since we did not change the contents, they are the same, but we use the location pointer to reexamine the currently opened word. Similarly, the programmer may use the period (.) as an arithmetic expression component, such as

```
.+5         SOJGE 2,ADR+3
```

DDT's location pointer is set to new value by the / command when immediately preceded by an address. For example,

```
201/        0
```

sets the location pointer to 201. If the user types / without typing an address, the contents of the location addressed in the last typeout are typed.

```
667/        MOVE 1,6        /       0
./          MOVE 1,6
```

Location 667 contains the instruction MOVE 1.6. The second slash displays the contents of Accumulator 6, which is zero. This does not change the location pointer, which is still pointing to location 667.

```
ADR/        MOVE A,CC1      /       ADD 2,SUM+7
```

It should also be noted that the spaces, which occur after DDT completes the typing of the contents of ADR, are automatically produced by DDT, not the user.

The left square bracket ([)[1] has the same effect as the slash (the address immediately preceding the [will be opened). However, [forces the typeout to be in numbers of the current radix.

```
ADR[ 11  (OCTAL)

ADR] 9.  (DECIMAL)
```

The right bracket (])[1] has the same effect as the slash except that it forces the typeout to be in symbolic instructions.

```
ADR+23 ]   MOVE 15,LIST+2
```

The exclamation point (!) works like the slash except that it suppresses type out of contents of locations until either /, or [, or] is typed by the user. The LINE FEED (↓) commands DDT to type out the contents of ADR+1.

```
ADR!        MOVE AC,555↓      (1)

ADR+1!↙                       (2)

ADR/      MOVE AC,555         (3)
```

Thus, in step (1) of the example the contents of ADR are not typed out, but the address is opened to modification and MOVE AC,555 has been typed in by the user.

Step (2) of the example shows that the location pointer has been incremented by one and the contents of ADR+1 are not typed out. This is because the exclamation point is still in effect and will continue to take effect until /, [, or] is typed in by the user. In this case, the slash terminates the effect of the exclamation point.

Step (3) shows that the modification (MOVE AC,555) of ADR typed in Step (1) has been accomplished.

3.2 CHANGING THE CONTENTS OF A WORD

After a word is opened, its contents can be changed by typing the new contents following the type out by DDT, followed by a carriage return. For example,

```
ADR/        MOVE A,CC1        MOVE A,CC2↙
```

[1] On Teletype Models 33 and 35 the left square bracket ([) is produced by holding the SHIFT key down and striking the K key. The right square bracket (]), is produced by holding the SHIFT key down and striking the M key.

The carriage return closes the open word, but does not move the location pointer. A LINE FEED (↓) command could also be used to make this modification. A LINE FEED causes a carriage return, adds one to DDT's location counter (moves the pointer), types out the resulting address and the contents of the new address. Thus, if we conclude our last example with a LINE FEED

```
ADR/        MOVE A,CC1        MOVE A,CC2↓
ADR+1/      ADD 3,CC3
```

ADR+1 is now open, and may be modified by the user.

The vertical arrow (↑)[1] works similarly, except that one is subtracted from the location pointer. The open word is closed (modified if a change is given) and the new address and contents are typed out.

```
ADR+1/      ADD3,CC3↑
ADR/        MOVE A,CC2
```

Since the vertical arrow subtracts one from the pointer, the resulting address is ADR, and the contents now show the change made in the previous example.

3.3 INSERTING A CHANGE, AND EXAMINING THE CONTENTS OF THE LAST TYPED ADDRESS

The horizontal tab (→|) causes a carriage-return line feed, then sets the location pointer to the last address typed (the new address if a modification was made) of the instruction in the register just closed. Then DDT types this new address, followed by a slash and the contents of that location, as shown below.

```
ADR5/       JRST ADR1 JRST ADR →|
ADR/        MOVEM 3,CC2 →|
CC2/        666
```

The backslash (\)[2] opens the word at the last address typed and types out the contents. However, backslash does not change the location pointer. The backslash closes the previously opened word and causes it to be modified if a new quantity has been typed in.

```
ADR/        MOVE A,CC2 JRST X\        MOVE AC,3
```

---

[1] ↑ is produced by SHIFT-N on Teletype Models 33 and 35. The backspace key may be used instead of ↑ on Teletype Model 37.

[2] \ is produced by SHIFT-L on Teletype Models 33 and 35.

The use of the backslash accomplishes two things. First it changes ADR by replacing its contents with JRST X. Second, the backslash causes DDT to type out the contents of X, namely, MOVE AC,3. The location pointer continues to point to ADR, but now location X is open and may be modified if desired.

If the line-feed control character and the vertical arrow were used in conjunction with the backslash, the results would be as follows.

```
ADR/      MOVEM B,CC2      MOVE A,CC1\ 105776↓
ADR+1/      MOVE A,C↑
ADR/      MOVE A,CC1     \ 105776
```

The following is a summary in table form of these special control characters and their corresponding functions. For example, the chart shows that the forward slash (/) will examine the contents of an address, type out in the current mode, open the address, change the location pointer to the address just opened, but it does not cause a new quantity to be inserted in that address.

Table 3-1
Special Character Functions

COMMAND CHARACTER	TYPE OUT CONTENTS	MODE	ADDRESS OPENED	CHANGE LOCATION POINTER	INSERT NEW QTY IF NEW QTY HAS BEEN TYPED
/	Yes	Current	Yes	Yes[1]	No
[	Yes	Numeric			
]	Yes	Symbolic			
!	No	None			
\	Yes[2]	Current	Yes	No	Yes
TAB (→|)	Yes[2]	Current	Yes	Yes	Yes
↑ or backspace	Yes[2]	Current	Yes	Yes (.-1)	Yes
Line-feed (↓)	Yes[2]	Current	Yes	Yes (.+1)	Yes
Carriage return (↲)	No	None	No (closes)	No	Yes

[1] If a user-typed quantity preceded.

[2] If ! has not suppressed typeout.

A ? typed by DDT when examining a location indicates that the address of the location is outside the user's assigned memory area. A ? typed when depositing indicates that the location cannot be written in, because it is either outside the assigned memory area or inside a write-protected memory segment.

3.4 STARTING THE PROGRAM

The program is started by typing

```
$G
```

This starts the program with the instruction beginning at the user's previously specified starting address taken from location JOBSA. The programmer may start at any other instruction by typing the address of that instruction followed by $G. For example,

```
4000$G    OR    ADR+5$G
```

starts the program at the instruction stored at location 4000 or, in the second part, at the symbolic address ADR+5. The start command may also be used to restart from breakpoints when the user does not wish to proceed to the next instruction.

3.5 RETYPING IN MODES OTHER THAN PREVAILING OR TEMPORARY

Each of the following commands specifies the mode in which DDT should immediately retype the last expression typed by DDT or the user. Neither the temporary nor the prevailing mode is altered.

3.5.1 Type Out Numeric

Although DDT is initialized to type out in symbolic mode, it is often useful to change to numeric typeout. When the programmer types the equal sign (=), the last expression typed is retyped by DDT in the current radix (initially octal). This is useful when a symbolic typeout is meaningless. Since this usually indicates that numeric data is stored in that word, the user can verify this by typing = and checking the value.

3.5.2 Type Out Symbolic

If a typeout is numeric, and the user wants to examine it in symbolic mode, he types the left arrow (←). The last typed quantity is retyped as a symbolic instruction. The address mode is determined by $A or $R.

3.5.3 Type Out in Current Mode

To retype a typeout in the current mode, the user types a semicolon (;). This may be used, for example, if the user has changed the typeout mode. For example,

```
TEXT/        A'DM    1,342212 (10)    5T;ABCDE
```

3.6 SYMBOLS

Before DDT commands can be used to reference local symbols in the program Symbol Table, the user should type the program name as specified in the MACRO-10 TITLE statement, or the FORTRAN IV SUBROUTINE or FUNCTION statement, followed by an ALTMODE and a colon. For example,

```
MAIN$:
```

makes the local symbols in the program called MAIN available. Since the user can debug several related subroutines simultaneously, reference to several independent symbol tables is permitted, each of which may use the same local symbols with different values. DDT allows the user to reference unique local symbols in other programs without respecifying the program name with $: (see Section 5.6.2). However, to access a local symbol that is used in several programs, the user must specify the program name to remove the ambiguity. Global symbols, such as those specified in MACRO-10 INTERNAL statements, may always be referenced.

The user may insert (or redefine) a symbol in the symbol table by typing the symbol, followed by a colon. The symbol will have a value equal to the address of the location pointer (.).

```
X/      ADD1 3,N  TAG:
```

causes TAG to be defined with the same value as X. All user defined symbols are global.

The user may also directly assign a value to a symbol by typing the value, a left angle bracket (<) and the symbol, terminated by a colon. This is the equivalent of a MACRO-10 direct assignment statement. Some examples are:

```
707<CONS:       12,1E+2<NUMB:

27<X:           101<MIL:
```

3.7 TYPING IN

To change or modify the contents of a word, the user may type symbolic instructions, numbers, and text characters. Type-ins are interpreted by DDT in context. That is, DDT tests the data typed in to determine whether it is to be interpreted as an instruction, a number (octal or decimal), or text. Typeout mode settings, such as $S, $C, and $nR, do not affect typed input.

The user may type the following:

1. Symbolic instructions
2. Numbers
   a. Octal integers
   b. Fixed-point decimal integers

c. Floating-point decimal mixed numbers

3. Text

   a. Up to five ASCII characters, left justified in a word

   b. Up to six SIXBIT characters, left justified in a word

   c. A single ASCII character, right justified in a word

   d. A single SIXBIT character, right justified in a word

4. Symbols

   Anything that is not a number or text is interpreted by DDT as a symbol.

3.7.1 Typing In Symbolic Instructions

In general, a symbolic instruction is written for insertion by DDT, in the same way the instruction is written as a MACRO-10 source program statement. For example:

```
X/      Ø       ADD     AC1,DATE
```

where a space terminates the instruction field, and a comma terminates the accumulator field. For example:

1. In DDT, the operation code determines the interpretation of the accumulator file. If an I/O instruction is used, DDT inserts the I/O device number in the correct place, and

2. Indirect and indexed addresses are written, as in MACRO-10 statements, where @ precedes the address to set the indirect bit, and the index register specified follows in parentheses.

   ```
   X/      Ø       ADD     4,@NUM(17)
   ```

   To type in two 18-bit halfwords, the left and right expressions are separated by two commas. For example:

   ```
   X/      Ø     A,,B
   ```

   This is similar to the MACRO-10 statement

   ```
   XWD     A,B
   ```

3.7.2 Typing In Numbers

A typed-in number is interpreted by DDT as octal if it does not contain a decimal point. The following examples are octal type-ins:

```
1234      -10101

772       777777777777
```

Fixed-point decimal integers must contain a decimal point with no digits following.

```
1234.    -99.    877.
```

Floating-point numbers may be written in two formats: with a decimal point and a digit following the decimal point

```
101.1   1234.5   999.0   -2.71828
```

or as in MACRO-10, with E indicating exponentiation

```
12.0E+2  77.3E+5  12.34E2  31.4159E-1
```

3.7.3 Typing In Text Characters

To type in up to five ASCII characters, left justified in an opened word, the user types a quotation mark, followed by any printing delimiting character, then the text characters, and terminated by the delimiting character. The following examples are legal:

```
"/TEXT/ "ABCDEFA
```

In these cases, / and A are the delimiting characters.

To type in up to six SIXBIT characters, left justified in an opened word, the user types ALTMODE quotation mark ($"), followed by any delimiting character, then the text characters, and terminated by repeating the delimiting character. Lower case letters are converted to upper case. Characters outside the SIXBIT set are illegal, and DDT types a question mark. The two examples below are SIXBIT type ins.

```
$" /DIVIDE/        $" EXXXXXXE
```

To type in a single ASCII character, right justified in an opened word, the user types a quotation mark, followed by a single ASCII text character, then by an ALTMODE.

```
"Q$   "/$   "?$
```

To type in a single SIXBIT character, right justified in an opened word, the user types an ALTMODE, followed by a quotation mark, a single SIXBIT text character and terminated by an ALTMODE.

```
$"Q$   $"M$   $"$$
```

3.7.4 Arithmetic Expressions

Numbers and symbols may be combined into expressions using the following characters to indicate arithmetic operations.

+ The plus sign means 2's complement integer addition.

- The minus sign means 2's complement integer subtraction.

* The asterisk means integer multiplication.

' The single quote means integer division with any remainder discarded. (The slash has another function.)

Symbols and numbers are combined by +, –, *, ' to form expressions. Examples:

```
6+2
S'2.51+BASE
2*3+1
.
```

3.8 DELETE

Any partially typed command may be deleted by pressing the RUBOUT or DELETE key. This causes DDT to ignore any preceding (unexecuted) partial command and DDT types XXX. The correct command may then be retyped.

3.9 ERROR MESSAGES

If the user types an undefined symbol which cannot be interpreted by DDT, U is typed back. If an illegal DDT command is typed, ? is typed back. Examining or depositing into a location outside the user's assigned memory area causes DDT to type a ?. Depositing in a write-protected high memory segment also results in a ? typeout.

3.10 UPPER AND LOWER CASE

DDT will accept alphabetic input in either upper or lower case. Lower case letters are internally converted to upper case, except when inputting text where they are taken literally as explained in Section 3.7.3.

DDT output is in upper case, except for text which is taken literally.

CHAPTER 4

MORE DDT-10 COMMANDS

This chapter describes other type-out modes, conditional breakpoints, searches and additional features. Commands are available to change modes from the initial settings so that numeric data can be typed out in a radix chosen by the user, in floating-point format, in RADIX50 format, as halfwords (two addresses) and as bytes of any size. The contents of a storage word may also be typed out as 7-bit DECsystem-10 ASCII text, or SIXBIT text characters. (See MACRO-10 Manual, Appendix C.)

Searches can be made in any part of the program for any word, not-word (inequality), or effective address. The user specifies the instruction or data to be searched for and the limits of the search.

Breakpoints can be set conditionally, so that a program stop occurs if the condition is satisfied. In addition, a counter can be set up allowing the user to specify the number of times a breakpoint is passed before a program stop occurs.

4.1 CHANGING THE OUTPUT RADIX

Any radix (>1) may be set by typing $nR, where n is the radix for the next typeout only, and n is interpreted by DDT as a decimal value. The radix is permanently changed when the double ALTMODE is used in the command $$nR. To change the type-out radix permanently to decimal, the user types,

```
$$10R
```

When the output radix is decimal, DDT follows all numbers with a point.

4.2 TYPE-OUT MODES

When DDT-10 is loaded, the type-out modes are initialized to produce symbolic instructions with addresses relative to symbolic locations. For numeric typeouts, the radix is initially set to octal.

These modes may be changed by the user. The duration, or lasting effect of a type-out mode change is also set by the user. Prevailing modes, which are semipermanent, are preceded by two ALTMODES. Temporary modes are preceded by a single ALTMODE. In addition, some mode changes effect only one typeout, such as the equal sign, which causes DDT to retype the last typed quantity in numeric mode.

In general, prevailing modes are changed by replacing them with another prevailing mode or by reinitializing the system. Temporary modes remain in effect until the user types a carriage return (↵), or re-enters DDT. One-time modes apply only to a single typeout.

4.2.1 Primary Type-out Modes

$S (OR $$S)

Type out symbolic instructions. The address part interpretation is set by $R or $A.

```
$S   ADR/       ADD    AC1,TABLE+3
```

$A (OR $$A)

Type out the address parts of symbolic instructions, and both addresses when the mode is halfword, as absolute numbers in the current radix.

```
$A   ADR/       ADD   4002
```

$R (OR $$R)

Type out addresses as relative addresses.

$C (OR $$C)

Type out constants; i.e., as numbers in the current radix.

```
$C   ABLE/      254111,,4050
```

If the output radix is octal and the left half is not 0, the word will be divided into halves separated by commas.

$F (OR $$F)

Type out the contents of storage words as floating point numbers.

```
$F   X/       0.07516230E-45
```

Unnormalized numbers are typed out as signed decimal integers.

$T (OR $$T)

Type out as 7-bit ASCII text characters. Left-justified characters are assumed unless the leftmost character is null. If the leftmost character is null, then right-justified characters are assumed.

```
$T   REX/       ABCDE
```

$6T (OR $$6T)

Type out as SIXBIT text characters.

```
$6T   REX/      ABCDEF
```

$5T (OR $$5T)

Type out symbols in radix 50 mode. (See MACRO-10 Manual, Chapter 6.)

```
$5T   13774/      4 CREF=40003,,261556
```

$H (OR $$H) This command causes the typeout to be in halfwords, the left half separated from the right half by double commas, The address mode interpretation is determined by $R or $A.

```
$A   $H   Z/        4503,,4502

$R   $H   Z/        TABL+14,,TABL+13
```

$NO (OR $$NO) Type out in n-bit bytes, where n is decimal. (Use the letter O, not zero.)

```
$60 BYTS/       22,23, 1, 73, 5, 46
```

As in all DDT typeouts, leading zeros are suppressed.

4.3 BREAKPOINTS

4.3.1 Setting Breakpoints

The programmer can automatically stop his program at strategic points by setting up to eight breakpoints. Breakpoints may be set before the debugging run is started, or during another breakpoint stop. To set a breakpoint, the programmer types the symbolic or absolute address of the word at the location at which he wants the program to stop, followed by $B. For example, to stop when location 4002 is reached, he types,

```
4002$B
```

If all eight breakpoints are in use, DDT will type a question mark. The user may assign breakpoint numbers when he sets a breakpoint by typing ADR $nB, where n is the breakpoint number (1<n<8). For example,

```
SYM$3B   ADR$7B
```

If n is not entered DDT will assign 1 through 8 in sequence. In the previous example, when ADR is reached, DDT types,

```
$7B>>ADR
```

indicating that the break has occurred at location ADR, and breakpoint No. 7 was encountered. The break always occurs before the instruction at the breakpoint address is executed.

If the instruction at the breakpoint location is executed by an XCT instruction, the typeout will show the address of the XCT instruction, not the location of the breakpoint. The program stops at each breakpoint address, and the programmer can then type other commands to examine and debug his program.

When the programmer sets a breakpoint, he may request that the contents of a word be typed out when a breakpoint is reached. To do this, the address of the word to be examined is inserted, followed by two commas, before the breakpoint address.

```
X,,4002$2B
```

When address 4002 is reached, DDT types out:

```
$2B>>4002   X/        ADD   AC,Y+2
```

where ADD AC, Y+2 is the contents of X. Location X is left open at this point. Location 0 may not be typed out in this way because a zero argument implies no typeout.

4.3.2 Removing Breakpoints

The user may remove a breakpoint by typing:

```
n$-B
```

where n is the number of the breakpoint to be removed. Therefore:

```
0$2B
```

removes the second breakpoint. All assigned breakpoints are removed by typing

```
$B
```

The user may reassign a breakpoint. If he has set breakpoint No. 2 at location ADR (ADR$2B), he may reassign No. 2 to ADR+1 by typing ADR+1$2B.

4.3.3 Restrictions for Breakpoints

Breakpoints may not be set on instructions that are:

1. Modified by the program
2. Used as data or literals
3. Used as part of an indirect addressing chain
4. The user mode monitor command, INIT

   A breakpoint at any other monitor command will operate correctly, except that if the monitor command is in error, the monitor will type out an error and the Program Counter, but the Program Counter will be Internal to DDT and meaningless to the user.

5. A breakpoint may not be assigned to accumulator 0.

4.3.4 Restarting After a Breakpoint Stop

To resume the program after stopping at a breakpoint, the user types the proceed command,

```
$P
```

The program is restarted by executing the instruction at the location where the break occurred. If the user types n$P, this breakpoint will be passed n-1 times before a break can occur; the break will occur the nth time. If n is not specified, it is assumed to be one. If the user proceeds by typing $$P (or n$$P), the program will proceed automatically when the program breaks again. If DDT encounters an XCT loop or the monitor command INIT when proceeding, a question mark will be typed.

Alternatively, the user may restart at any location by typing the start command,

```
ADR$G
```

where ADR is any program address, or $G, which restarts at the previously specified starting address in location JOBSA.

4.3.5 Automatic Restarts from Breakpoints

If the user requests DDT to type out the contents of a word and then continue program execution without stopping, he types two ALTMODES when specifying the breakpoint address.

```
AC,,ADR$$B
```

When ADR is encountered, the contents of AC are typed out and program execution continues. To get out of the automatic proceed mode, type any Teletype key during the typeout, and then remove the breakpoint or reassign it with a single ALTMODE. It may be necessary to use ↑C and DDT to get back to DDT to remove or reassign the breakpoint.

4.3.6 Checking Breakpoint Status

The user may determine the status of a breakpoint by examining locations $nB, $nB+1, and $nB+2.

$nB contains the address of the breakpoint in the right half and the address of the location to be examined in the left half. If both halves equal zero, the breakpoint is not in use.

$nB contains the conditional breakpoint instruction. (See Paragraph 4.3.7.)

$nB+2 contains the proceed count.

4.3.7 Conditional Breakpoints

Breakpoints may be set up conditionally in two ways. The user may provide his own instruction or subroutine to determine whether or not to stop, or he may set a proceed counter which must be equal to or less than zero in order for a break to occur.

When a breakpoint location is reached, DDT enters its breakpoint analysis routine consisting of five instructions.

```
SKIPE    $NB+1          ;Is the conditional break instruction 0?
XCT      $NB+1          ;No, execute conditional break instruction
SOSG     $NB+2          ;Decrement and text the proceed counter
JRST     break routine
JRST     proceed routine
```

If the contents of $nB+1 are zero (indicating that there is no conditional instruction) the proceed counter at $nB+2 is decremented and tested. If it is less than or equal to zero, a break occurs; if it is greater than zero the execution of the user's program proceeds with the instruction where the break occurred.

If the conditional break instruction is not zero, it is executed. If the instruction (or the closed subroutine) does not cause a program counter skip, the proceed counter is decremented and tested as above. If a program counter skip does occur, a break occurs. If the conditional instruction is a call to a closed subroutine which returns skipping over two instructions, execution of the user's program proceeds.

If the user wishes a break to occur based only on the conditional instruction, he should set the proceed counter to a large positive number so that the proceed counter will never reach zero.

4.3.7.1 Using the Proceed Counter – If the user wishes to proceed past a breakpoint a specified number of times, and then stop, he inserts the number of passes in $nB+2, which contains the proceed count.

The proceed counter may be set in two ways. The first way is by direct insertion. For example,

```
$NB+2/      0  20
```

sets the counter to 20. The second method is as follows. After stopping at a breakpoint, the proceed count may be set (or reset) by typing the count before the proceed command:

```
20$P
```

($P will proceed from the interrupted instruction sequence even if the breakpoint has been removed or reassigned.)

4.3.7.2 Using the Conditional Break Instruction – The user inserts a conditional instruction, or a call to a closed subroutine at $nB+1. For example,

```
$33+1/      0     CAIGE  ACC,15
```

or

```
$4B+1/      0     JSA    16,TEST
```

When the breakpoint is reached, this instruction or subroutine is executed. If the instruction does not skip or the subroutine returns to the next sequential location, the proceed counter is decremented and tested, as explained in Paragraph 4.3.7. If the instruction skips or the subroutine returns skipping over one instruction, the program breaks. If the subroutine causes a double skip return, the program proceeds with the instruction at the breakpoint address.

Examples of Conditional Breakpoints

If address 6700 is reached and DDT's No. 4 breakpoint registers are as follows:

```
$4B/       AC1,,6700

$4B+1/       CAIE  AC1,100

$4B+2/       200
```

AC1 contains 100, and DDT types

```
$4B>6700   AC1/   100
```

Since AC1 contains 100, the compare instruction skips and the program breaks. If AC1 did not contain 100, $4B+2 would be decremented by one and the user's program would continue running.

If the conditional break instruction transfers to a subroutine which, after the subroutine is executed, returns to the calling location +3, a break will never occur regardless of the proceed counter. Example: If the internal DDT breakpoint registers ($2B and $2B+1) have the following contents, a break would not occur unless accumulator 3 contains 100.

```
$2B/      ADR
$2B+1/      JSR TEST
TEST/     Ø
TEST+1/     AOS TEST      (contains PC when JSR to subroutine TEST
                          is made)
TEST+2/     CAIE 3,1ØØ
TEST+3/     AOS TEST
TEST+4/     JRST @ TEST
```

The subroutine TEST causes a double skip (the return is to the third instruction after the call) in DDT if accumulator 3 does not equal 100. A break will never occur at address ADR (regardless of the proceed counter) unless accumulator 3 contains 100.

4.3.8 Entering DDT from a Breakpoint

When a break occurs, the state of the user's program is saved, the JSR breakpoint instructions are removed, and the programmer's original instructions are restored to the breakpoint location. DDT types out the number of the breakpoint and a symbol indicating the reason for the break, > for the conditional break instruction, >> for the proceed counter and the address in the user's program where the break occurred.

Example: If address ADR is reached in the user's program and DDT's breakpoint registers contain:

```
$2B/      ADR
$2B+1/      Ø
$2B+2/      Ø          (proceed counter contains zero)
```

DDT stops the program and types,

```
$2B>>ADR
```

4.3.9 Single Instruction Proceed

To execute the next instruction, the user types

$X

which, without an argument, executes the instruction about to be executed after the last breakpoint or the last $X. After the instruction is executed, the PC is updated. (The breakpoint, however, is not moved.) After any number of $X's, $P will always proceed from where the single stepping left off. After executing the instruction, DDT prints out the contents of referenced locations. These are printed in floating point where appropriate. The modified flags are also printed out for JRSTF and JFCL. Then the next instruction is printed out (always in symbolic despite temporary or permanent output mode settings). A blank line is printed between the operands and the next instruction, provided that the instruction just executed was a skip or a jump that succeeded.

The following is a list of forms taken by the $X command:

1. n$X where n<2**27: performs $X n times, as above.
   where n is an instruction, the command performs as it always has.

2. n$$X where n 2**27: is the same as n$X, except that printout is suppressed for all but the last instruction executed.

3. $$X without an argument, perform $X indefinitely, without printing anything until the PC reaches either .+1 or .+2. This command is useful if one wants to execute a debugged subroutine.

NOTE

DDT looks for typein after each instruction in an n$X cycle. This procedure is followed by the search logic also.

Breakpoints are not in place during a $X.

In exec mode, DDT does not restore the RI system during $X.

4.4 SEARCHES

There are three types of searches: the word search, the not-word search, and the effective address search.

Searches can be done between limits. The format of the search command is:

	W	Word search
a<b>c$	N	Not-word search
	E	Effective address search

where:

a is the lower limit of the search; 0 is assumed if this argument and its delimiter are not present.

b is the upper limit of the search. The lower numbered end of the symbol table is assumed if this argument and its delimiter are not present.

c Is the quantity searched for.

The effective address search (E) will find and type out all locations where the effective address, following all indirect and index-register chains to a maximum length of 64(10) levels, equals the address being searched for.

Examples:

```
4517/<5000>X$E
INPUT<5002>700$E
```

Examples of DDT output, when searching for X in the above example, are as follows.

```
4517/     SETZM X
4721/     MOVE 2,X
5000/     MOVE 3, @ 4721  (Indirectly addresses X through address 4721)
```

The word search (W) and the not-word search (N) compare each storage word with the word being searched for in those bit positions where the mask, located at $M, has ones. The mask word contains all ones unless otherwise set by the user. If the comparison shows an equality, the word search types out the address and the contents of the register; if the comparison results in an inequality, the word search will type out nothing. The not-word search types nothing if an equality is reached. It types the contents of the register when the comparison is an inequality.

Examples:

```
INPT<INPT+10>NUM$W
INPT<INPT+10>0$N
```

$M/		This command types out the contents of the mask register, which is then open. The contents of the mask register are ordinarily all ones unless changed by the user.
N$M		Inserts n into the mask register.
Ø$M	FIRST<LAST>Ø$W	Lists a block of locations by setting the MASK to zero then performing a word search for zero.

4.5 MISCELLANEOUS COMMANDS

$Q		This symbol represents the last quantity typed.
ADR/	1ØØ,2ØØ	$Q puts back in ADR the quantity 100,,200. $Q+1 puts back in ADR the quantity 100,,201. $Q/ displays the contents of location 200. $Q+1/ displays the contents of location 201.
$$Q		This symbol represents the last quantity typed with the two halves of the word reversed. ($$Q was formerly $V.)
ADR/	1ØØ,,2ØØ	$$Q puts back in ADR the quantity 200,,100. $$Q+1 puts back in ADR the quantity 200,,101. $$Q/ displays the contents of location 100. $$Q+1/ displays the contents of location 101.
INST$X		This command causes the instruction inst to be executed.
JRST ADR$X		Starts the user's program at ADR.
FIRST<LAST$$Z		This command zeros the memory locations between the indicated FIRST and LAST address inclusively. If the first address is not present, location 0 is assumed. If the last address is not present, the location before the low-numbered end of the symbol table is assumed. Locations 20-137, DDT, and the symbol table are not zeroed.
$Y		This command causes a command file to be read and executed, in user mode, the default name for the command file is DSK:BATCH.DDT. The command string $"/NAME/$Y causes the file DSK:NAME.DDT to be interpreted. In exec mode, the command reads a command file from the paper tape reader.

When DDT is reading a command file, rubouts and the character immediately following a carriage return (assumed to be a linefeed) are ignored. Any sequence of DDT commands including $X, $G is legal.

The ? error message is given if

1. A lookup failure occurs on the command file, or
2. This command is not implemented.

CHAPTER 5

SYMBOLS AND DDT ASSEMBLY

A symbol is defined in DDT as a string of up to six letters and numbers including the special characters period (.), percent sign (%), and dollar sign ($). Characters after the sixth are ignored. A symbol must contain at least one letter. If a symbol contains numerals and only one letter, that letter must not be a B, D, or an E. These letters are reserved for binary-shifted and floating-point numbers.

Certain symbols can be referenced in one program from another. These symbols are called "global". Those which can only be referenced from within the same program are called "local" or "internal". Any symbol which has been defined as global by MACRO-10 (using the INTERNAL or ENTRY statements) will be considered as global by DDT-10 when it is referenced. FORTRAN subroutine entry points and common block names are globals. All symbols which the user defines via DDT are defined or redefined as global symbols.

The user may want to reference a local symbol within a particular program. In order to do this he should first type the program name followed by $:. Thus, if a user wishes to use a symbol local to program MIN, he types the command,

```
MIN$:
```

This command unlocks the symbol table associated with MIN. DDT allows the user to reference unique local symbols in other programs without respecifying the program name with $: (see Section 5.6.2). However, to access a local symbol that is used in several programs, the user must specify the program name to remove the ambiguity. The program name is that specified in the MACRO-10 TITLE statement. In FORTRAN, the program name is either MAIN., the name from the SUBROUTINE or FUNCTION statement, or DAT. for BLOCK DATA subprograms.

5.1 DEFINING SYMBOLS

There are two ways to assign a value to a symbol.

NUMERIC VALUE<SYMBOL:

This command puts SYMBOL into DDT-10's symbol table with a value equal to the specified NUMERIC VALUE. SYMBOL is any legal symbol defined or undefined.

Example:

```
305<XVAR:
```

XVAR has now been defined to have the value 305.

TAG: This command puts TAG into DDT-10's symbol table with a value equal to the address of the location pointer.

Example:

```
400/        ADD 2, 12012        X:
```

This puts the symbolic tag X into DDT-10's symbol table and sets X equal to 400, the address of the last register opened.

5.2 DELETING SYMBOLS

There are times when the user will want to restrict or eliminate the use of a certain few defined symbols. The following three ways give the user of DDT-10 these capabilities.

SYMBOL $$K — SYMBOL is killed (removed) in the user's symbol table. SYMBOL can no longer be used for input or output.

Example:

```
X$$K
```

This command removes the symbol X from the symbol table.

SYMBOL $K — This command prevents DDT from using this symbol for typeout; it can still be used for typein. For example, the user may have set the same numeric value to several different symbols. However, he does not wish certain symbol(s) to be typed out as addresses or accumulators.

```
X/        MOVE J,SAV J$K ←MOVE AC,SAV N$K  ← MOVE AC,SAV
```

Since the user does not wish J to be typed out as an accumulator, he types in J$K, followed by a left arrow to type out the contents of X again and MOVE N.SAV is typed out. He then repeats the above process until the desired result, namely AC, is typed out. Any further symbolic typeouts with the same number in the accumulator field of the instruction will type out as AC.

$D	The last symbol typed out by DDT has $K performed on it. The value of the last quantity output is then retyped automatically. For example,

```
A/       MOVE AC,LOC $D  MOVE AC,ABC+1
```

5.3 DDT ASSEMBLY

When improvising a program on-line to the DECsystem-10 on a terminal, the user will want to use symbols in his instructions in making up the program. In this and in other situations, undefined symbols may be used by following the symbol with the number sign (#). The symbol will be remembered by DDT from then on. Until the symbol is specifically defined by the use of a colon, the value of the symbol is taken to be zero. Successive use of the undefined symbol causes DDT to type out #. Appending # to all subsequent uses of the symbol enables the user to readily identify undefined (not yet defined by a colon) symbols. When an undefined symbol is finally defined, all previously tagged (#) occurrences of the symbol will be filled in.

Example:

```
MOVE 2,VALUE#
```

VALUE is now remembered by DDT and may be used further without the user appending the #. If subsequent instructions are given involving VALUE, DDT appends a # automatically to that symbol. Thus VALUE will always appear as VALUE followed by the # (until VALUE is defined).

Example:

```
START!      MOVE 2,VALUE#↓      (user types the #)
START+1!      ADDI 2,VALUE↓
START+2!      MOVEM 2,VALUE↓
#                                 (DDT types #)
START+3!      JRST VALUE+#1↓      (DDT types # after the plus sign because
                                  only at that point does DDT realize the
                                  symbol VALUE is complete.)
START+4!
```

Undefined symbols can be used only in operations involving addition or subtraction. The undefined symbols may be used only in the address file.

Example:

```
MOVEI 2,3*UNDEF#
```

This is an illegal operation – multiplication with a symbolic tag (UNDEF) which has not previously been defined.

The question mark (?) is a command to DDT to list all undefined symbols that have been used in DDT up to that point in the program.

Example:

```
?
VALUE
UNDEF
```

5.4 FIELD SEPARATORS

The storage word is considered by DDT to consist of three fields: the 36-bit wholeword field; the accumulator or I/O device field; and the address field. Expressions are combined in these three fields by two operators:

Space	The space adds the expression immediately preceding it (normally an op code) into the storage word being formed. It also sets a flag so that the expression going into the address field is truncated to the rightmost 18 bits.
Single Comma	The comma does three things: the left half of the expression is added into the storage word; the right half is shifted left 23 bits (into the accumulator field) and added into the storage word. If the leftmost three bits of the storage word are ones, the comma shifts the right half expression left one more place (I/O instructions thus shift device numbers into the device field). The comma also sets the flag to truncate addresses to 18 bits.
Double Comma	Double commas are used to separate the left and right halves of a word with contents expressed in halfword mode.

The address field expression is terminated by any word termination command or character.

5.5 EXPRESSION EVALUATION

Parentheses are used to denote an index field or to interchange the left and right halves of the expression inside the parentheses. DDT handles this by the following generalized procedure.

A left parenthesis stores the status of the storage-word assembler on the pushdown list and re-initializes the assembler to form a new storage word. A right parenthesis terminates the storage word and swaps its two halves to form the result inside the parentheses. This result is treated in one of two ways:

1. If +, –, ', or * immediately precede the left parenthesis, the expression is treated as a term in the larger expression being assembled and therefore may be truncated to 18 bits if part of the address field.

2. If +, –, ', or * did not immediately precede the left parenthesis, this swapped quantity is added into the storage word.

Parentheses may be nested to form subexpressions, to specify the left half of an expression, or to swap the left half of an expression into the right half.

5.6 SYMBOL EVALUATION

5.6.1 Order of Symbol Table Search

DDT references two symbol tables:

1. A built-in operation table containing the machine language instructions and monitor UUOs (e.g., MOVE, JRST, and INIT), and

2. A symbol table constructed by LOADER during the loading process, containing all the user-defined symbols.

When a user types a symbol into DDT, which must be converted into a binary value, DDT has two places to look for the symbol. If the expression (see Section 5.5) constructed has a zero value (the normal case when typing in the operation code of an instruction such as the JRST part of a JRST ADDRESS instruction), DDT looks for the symbol first in its internal operation table, and then, if the symbol is not found, in the LOADER constructed symbol table. If the expression constructed is non-zero, DDT searches the LOADER constructed table first, and then the internal operation table. This method of searching the table allows instructions such as JRST JRST to work correctly (the first JRST is an operation code, and the second JRST is a user-defined address location).

5.6.2 Order of Symbol Table Search for Symbol Evaluation

When DDT searches the LOADER constructed symbol table to evaluate a symbol typed in, it begins the search by looking through the symbols specified by <program name>$: (see Section 2.5). DDT searches the table in the following order:

1. Looks for the symbol as a local or global symbol in the currently unlocked (by $:) program symbols.

2. Looks for the symbol as a global symbol anywhere in the symbol table.

3. Looks for the symbol as a local symbol in the symbol table of one and only one program.

4. Looks for the symbol as a local symbol that appears in the symbol table of more than one program, but with the same value in each table. (If the symbol appears with different values in different tables, it will not be recognized as defined because there is no way to solve the ambiguity.)

5. If all the above fail, the symbol is undefined unless it appears in the internal operation table of the DDT.

Fortunately, the searching is accomplished with a single pass over the symbol table.

If one of the several identical local symbols (in step 4) is redefined, it becomes a global, and the symbol is then found at either step 1. or step 2.

This procedure relaxes the requirement of Sections 2.5, 3.6, and the beginning of Chapter 5 on the user of $: to unlock local symbols.

5.7 SPECIAL SYMBOLS

The @ sign sets the indirect bit in the storage word being formed.

Example:

```
MOVE AC,@X
```

5.8 BINARY VALUE INTERPRETATION

When DDT is typing the symbolic equivalent of a binary word or address, it looks for the symbol with a value that best matches the binary. DDT looks through the symbol values in the following order:

1. Searches the symbols of the currently unlocked (by $:) program for a local or global symbol with a value that exactly matches the binary to be interpreted.

2. Searches for a global symbol outside the currently locked program with a value that exactly matches the binary to be interpreted.

3. Searches all the other local symbol tables for one or more entries with values that match the binary to be interpreted. If more than one symbolic equivalent is found, the DDT does not use any of them but goes on to step 4. If exactly one

symbolic equivalent is found (this includes the case of the same symbol with the same value in more than one local symbol table), then this symbol is used. However, the symbol has a # appended to it to warn the user that this symbol might have a different value in some other local symbol table.

4. Searches the currently unlocked program symbols for a local symbol, and searches the entire symbol table for a global symbol, with the value closest to but less than the binary to be interpreted. The closest symbol is then used for typeout if it is not more than 64 smaller than the binary being interpreted.

If a usable symbol is not found in any of the above steps, the binary is typed out as an integer in the current output radix.

The purpose of this complicated procedure is to output the best symbol without forcing the user to continually respecify the program symbol table names by using $:.

CHAPTER 6

PAPER TAPE[1]

6.1 PAPER TAPE CONTROL

The following commands are used in paper tape control:

$L	This command causes DDT to punch a RIM10B loader on paper tape RIM10B loader. (See MACRO-10 Manual, Chapter 6.) Thus, if the user wishes to punch out a program on paper tape he gives a $L command first in order to get a loader punched on the same tape as the program. Later when the user wishes to read in the program from the paper tape, the hardware READ-IN feature will load the RIM10B loader into the accumulators and then the program will be loaded by the RIM10B loader. (See Figure 6-1.)
FIRST<LAST TAPE (2)	This command punches out checksummed blocks in RIM10B format on paper tape from consecutive locations between FIRST and LAST address inclusively. For example, this command will punch out a program existing in core memory in its present state of checkout for later use. Example: 4000<23000 TAPE
FIRST<LAST ↑B TAPE	This command is similar to the preceding command, except that locations whose contents are zero are not punched out whenever more than two consecutive zeros are detected.
ADR$J	This command punches a 2-word block that causes a transfer address ADR after the preceding program has been loaded from paper tape. If ADR is not present, a JRST 4, DDT is punched as the first word.

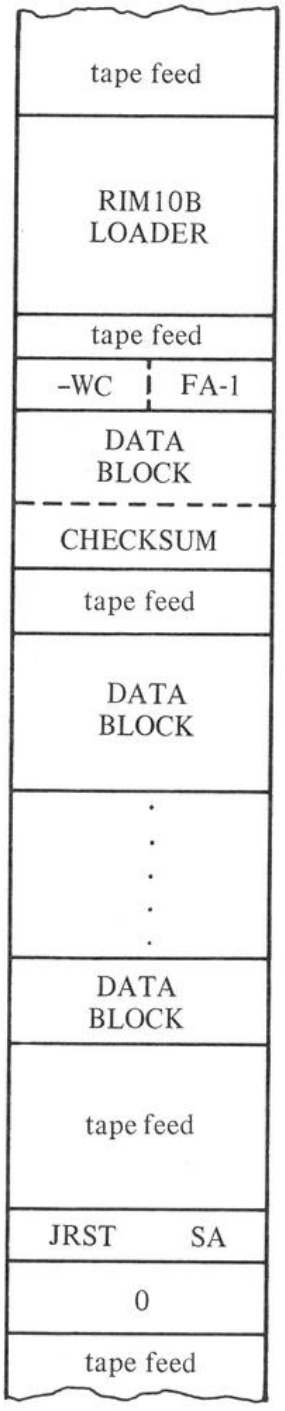
[1] The paper tape functions are not available in the timesharing user mode version of DDT.

[2] TAPE is a single control key on the teletypewriter and is identical to ↑R (control-R).

The following succession of steps will punch a program on paper tape ready to be used as an independent entity.

1. $L
2. 5000<2000 TAPE (2)
3. 6000J (Transfer to address 6000 after program is loaded.)

Typed in:



Beginning of Tape

Checksum includes pointer word
WC = word count

transfer block
SA = starting address

Figure 6-1
RIM10B Block Format

APPENDIX A

SUMMARY OF DDT FUNCTIONS

A.1 TYPE-OUT MODES

The following are used to set the type-out modes:

	Type	Sample Output(s)
Symbolic instructions	$S	ADD 4, TAG+1 ADD 4, 400002
Numeric, in current radix	$C	69, 1 5
Floating point	$F	0.125E-3
7-bit ASCII text	$T	PQRST
SIXBIT text	$6T	TSRQP0
RADIX50	$5T	4 DDTEND
Halfwords, two addresses	$H	400002,,400005 X+1,,X+4
Bytes (of n bits each)	$NO	$20 COULD YIELD 0,14,237,123,0

A.2 ADDRESS MODES

The following are used to set the address mode for typeout of symbolic instructions and halfwords (see examples above):

Relative to symbolic address	$R	TAG+1
Absolute numeric address	$A	400005

A.3 RADIX CHANGE

The following is used to change the radix of numeric type-outs:

to n (for r≥2): $NR $2R COULD YIELD
1101011000020102000000020111013112

A.4 PREVAILING VS. TEMPORARY MODES

The following are used in prevailing vs. temporary modes:

To set a temporary type-out or address mode or a temporary radix as shown in the commands above, type	$$	$C $10R
To set a prevailing type-out or address mode on a prevailing radix, in the commands above, substitute	$	$$C $$10R
To terminate temporary modes and revert to prevailing modes, type a carriage return	↵	
Initial prevailing (and temporary) modes are		$$S $$R $$8R

A.5 STORAGE WORDS

The following are used to examine storage words:

To open and examine the contents of any address in current type-out mode	adr/	LOC/ 254020,,DDTEND
To open a word, but inhibit the type out	adr!	LOC!
To open and examine a word as a number in the current radix	adr[	LOC[254020,,3454
To open and examine a word as a symbolic instruction	adr]	LOC] JRST @DDTEND
To retype the last quantity typed (particularly used after changing the current type-out mode)	;	$60; 25,40,20,00,34,54 $6T; 5@0 <L

A.6 RELATED STORAGE WORD

The following are used to examine related storage words:

To close the current open word (making any modification typed in) and to open the following related words, examining them in the current type-out mode:	
To examine ADR-1	↑ (or backspace on some hard-copy terminals)
To examine the contents of the location specified by the address of the last quantity typed, and to set the location pointer to this address	→| (TAB)
To examine the contents of address of last quantity typed, but not change the location pointer	\(backs|ash)
To close the currently open word, without opening a new word, and revert to permanent type-out modes	↵(carr|age return)

A.7 RETYPING IN MODES OTHER THAN PREVAILING OR TEMPORARY

Each of the following commands specifies the mode in which DDT should immediately retype the last expression typed by DDT or the user. Neither the temporary nor the prevailing mode is altered.

To repeat the last typeout as a number in the current radix	=
To repeat the last typeout as a symbolic instruction (the address part is determined by $A or $R)	←
To type out, in the current type-out mode, the contents of the location specified by the address in the open instruction word, and to open that location, but not move the location pointer	/

To type out, as a number, the contents of the location specified by the open instruction word and to open that location, but not move the location pointer	[	
To type out, as a symbolic instruction, the contents of the location specified by the open instruction word, and to open that word, but not move the location pointer	]	
To examine ADR+1	(line feed)	

A.8 TYPING IN

Current type-out modes do not affect typing in; instead, the following are performed:

To type in a symbolic instruction	ADD AC1,@DATE(17)	
To type in half words, separate the left and right halves by two commas	402,,403	
To type in octal values	1234	
To type in a fixed-point decimal integer	99.	
To type in a floating-point number	101.11 77.0E+2	
To type in up to five 7-bit PDP-10 ASCII characters, left justified, delimited by any printing character	"/ABCDE/	(/ is delimiter)
To type in one PDP-10 ASCII character, right justified	"A	($ must be ALTMODE)
To type in up to six SIX-BIT characters, left justified, delimited by any printing characters	$"ABCDEFGA	(A is delimiter)

To type in one SIXBIT character, right justified	$"C$	($ must be ALTMODE)

A.9 SYMBOLS

The following are DDT symbols:

To permit reference to local symbols within a program titled name	name$	MAIN,$:
To insert or redefine a symbol in the symbol table and give it the value n	n<symbol	14<TABL3:
To insert or redefine a symbol in the symbol table, and give it a value equal to the location pointer (.)	symbol:	SYM:
To delete a symbol from the symbol table	symbol ^$K	LPCT$$K
To kill a symbol for typeouts (but still permit it to be used for typing in)	symbol$K	TRITS$K
To perform $K on the last symbol typed out and then to retype the last quantity	$D	
To declare a symbol whose value is to be defined later	symbol#	JRST AJAX#
To type out a list of all undefined symbols (which were created by #)	?	

A.10 SPECIAL DDT SYMBOLS

The following are special DDT symbols:

To represent the address of the location pointer	. (point)
To represent the last quantity typed	$Q

To represent the last quantity typed, halves reversed	↑$Q
To read and execute a command file	↑Y
To represent the indirect address bit	@
To represent the address of the search mask register	↑M
To represent the address of the saved flags, etc. (see Appendix D)	↑I
To represent the pointers associated with the nth breakpoint	↑nB

A.11 ARITHMETIC OPERATORS

The following arithmetic operators are permitted in forming expressions:

Two's complement addition	+
Two's complement subtraction	-
Integer multiplication	*
Integer division (remainder discarded)	' (apostrophe)

A.12 FIELD DELIMITERS IN SYMBOLIC TYPE-INS

The following are field delimiters:

To delimit op-code name	one or more spaces	JRST SUBRTE
To delimit accumulator field	, (comma)	
To delimit two halfwords	left,,right	-6,,BEGIN-1
To delimit index register	( )	
To indicate indirect addressing	@	

A.13 BREAKPOINTS

The following are used for breakpoints:

To set a specific breakpoint n(1<n<8)	adr$nB	CAR$8B
To set the next unused breakpoint	adr$B	303$B
To set a breakpoint with automatic proceed	adr$$nB adr$$B	CAR$$8B 303$$B
To set a breakpoint which will automatically open and examine a specified address, x	x,,adr$nB x,,adr$B x,,adr$$nB x,,adr$$B	AC3,,Z+6$5B AC4,,ABLE$B AC3,,Z+6$$5B AC4,,ABLE$$B
To remove a specific breakpoint	0$nB	0$B
To remove all breakpoints	$B	$B
To check the status of breakpoint n	$nB/	
To proceed from a breakpoint	$P	$P
To set the proceed count and proceed	n$P	25$P
To proceed from a breakpoint and thereafter proceed automatically	$$P n$$P	$$P 25$$P

A.14 CONDITIONAL BREAKPOINTS

The following are used for conditional breakpoints:

To insert a conditional instruction (INST) or call a conditional routine, when breakpoint n is reached	$nB+1/ $2B+1/0	INST CAIE 3,100

If the conditional instruction does not cause a skip, the proceed counter is decremented and checked. If the proceed count <0, a break occurs

If the conditional instruction or subroutine causes one skip, a break occurs.

If the conditional instruction or subroutine causes two skips, execution of the program proceeds.

A.15 STARTING THE PROGRAM

The following commands are used to start the program:

To start at the starting address in JOBSA	$G	$G
To start, or continue, at a specified address	adr $G	LOC$G
To execute an instruction	inst $X	JRST 2, @JCBOPC$X returns to program after ↑C and DDT commands

A.16 SEARCHING

The following commands are used for searching:

To set a lower limit (a), an upper limit (b), a word to be searched for (c), and search for that word	a<b>c$W	200<250>0$W
To set limits and search for a not-word	a<b>c$N	351<731>0$N
To set limits and search for an effective address	a<b>c$E	421<471>LOC+6$E

To examine the mask used in searches (initially contains all ones)	$M/	$M/ -1
To insert another quantity n in the mask	n$M	777777777777$M

A.17 UNUSED FUNCTIONS

The following are unused:

$U

$V – formerly same as $$Q

A.18 ZEROING MEMORY

The following are used for zeroing memory:

To zero memory, except DDT, locations 20-137, and the symbol table	$$Z	
To zero memory locations FIRST through LAST inclusive	FIRST<LAST $$Z	

A.19 SPECIAL CHARACTERS

The following special characters are used in DDT typeouts:

Breakpoint stops

Break caused by conditional break instruction	>	
Break because proceed counter <0	>>	
Undefined symbol cannot be assembled	U	
Half-word type-outs	left,,right	401,,402
Unnormalized floating-point number	#1.234E+27	#1.234E+27

To indicate an integer is decimal. The decimal point is printed	$10R 77=63.
Illegal command	?
If all eight breakpoints have been assigned	?
RUBOUT echo	XXX

A.20 PAPER TAPE COMMANDS

The following commands are available only in EDDT:

To punch a RIM10B loader	$L
To punch checksummed data blocks where ADR1 is the first, and ADR2 is the last location of the data	ADR1<ADR2 TAPE
To punch data as above, except that more than two consecutive locations containing zeros are not punched.	ADR1<ADR2 $ TAPE (TAPE is ↑R)
To punch a one-word block to cause a transfer to adr after the preceding program has been loaded from paper tape	adr $J

APPENDIX B

EXECUTIVE MODE DEBUGGING (EDDT)

A special version of DDT, called EDDT, is available for debugging programs in the executive mode of the DECsystem-10. EDDT also runs in user mode under the monitor and performs the same debugging functions as user-mode DDT. EDDT requires somewhat more memory space than DDT; therefore, it is normally used only with hardware diagnostics and the monitor. All of the paper tape commands are available in EDDT (those in DDT are marked by an asterisk in Chapter 5). The paper tape I/O routines in EDDT are optional at assembly time.

EDDT is used to debug monitor programs, diagnostic programs, and other executive (or privileged) programs. EDDT performs its own I/O on a Teletype and controls the Priority Interrupt system. It does not check JOBREL for boundary limits as DDT does.

In EDDT, the symbol table pointer is in location 36 and the undefined-symbol table pointer is in location 32. If the NXM STOP switch is ON, the machine will hang up if nonexistent memory is referenced. If this happens, EDDT may be restarted by pressing START, or the CONTINUE switch may be pressed.

Stand-alone programs should initialize EDDT by placing the contents of .JBSYM (116) into location 36, and .JBUSY (117) into location 32.

The first address of EDDT is DDT; the last is DDTEND.

The $$Z command will not zero locations 20 through 37. (In the user mode version, $$Z does not zero locations 20 through 127. See Section 4.5.)

APPENDIX C

STORAGE MAP FOR USER MODE DDT

See Figure C-1. The permanent symbol table, which contains all DECsystem-10 instructions and monitor UUOs, is an integral part of DDT.

If the user's symbol table is overwritten DDT can still interpret all instructions and UUOs. It will not interpret I/O device mnemonics, internal $ symbols ($M, $I, $1B through $8B), DDT and DDTEND of the following:

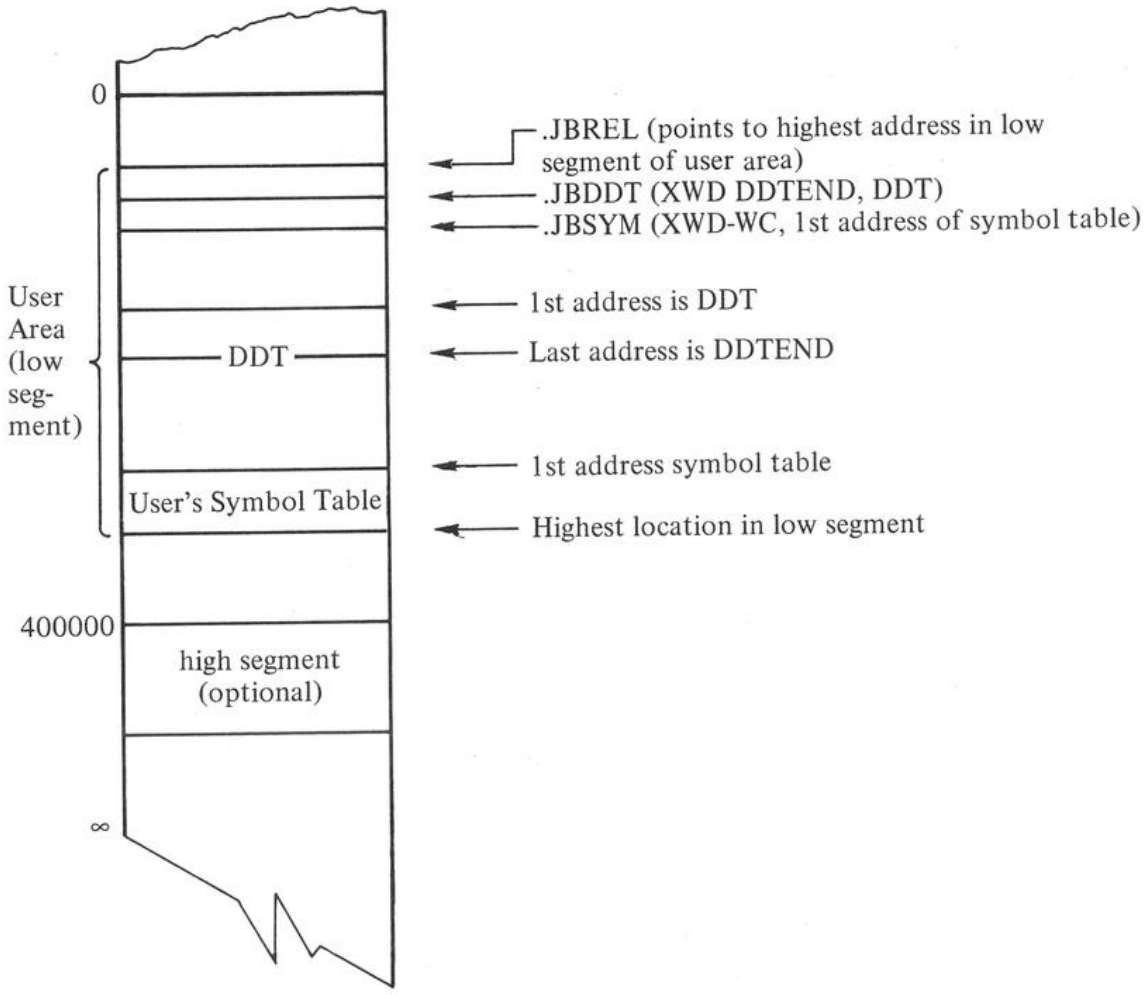


Figure C-1
Storage Map for User Mode DDT

APPENDIX D

OPERATING ENVIRONMENT

D.1 ENTERING AND LEAVING DDT

When control is transferred to DDT, the state of the machine is saved inside DDT:

1. The accumulators are saved.
2. The status of the priority interrupt system (the result of a CONI PI, $I) is stored in the right half of register $I.[1]
3. The central processor flags are saved in the left half of register $I.
4. The PI channels are turned off (by a CONO PI, @$I+1) if they have a bit in register $I+1.[1]
5. The Teletype PI channel is saved in the right half of register $I+2. The Teletype buffer is saved in the left half of $I+2 but can never be restored. The character in the output buffer will have been typed on the Teletype.[1]
6. If DDT was entered via ↑C↑C and the monitor DDT command, the old program counter word is saved in location JOBOPC.

When execution of a program is restarted, the following happens:

1. The accumulators are restored.
2. Those PI channels which were on (when DDT was entered) and which have a bit equal to 1 in register $I+1 are turned on.[1]

   (C($I)(R) C($I+1)(R))∨2000 PI SYSTEM

   (logical AND (∧), logical OR (∨))

3. The Teletype PI channel is restored.[1]

   0 TTI DONE TTI BUSY TTO BUSY

TTO done is set to 1 if either TTO busy or TTO done was on when DDT was entered. Otherwise 0 TTO done.

[1] Functions are not available in the timesharing user mode.

4. The processor flags are restored from the left half of register $I.

5. To return to a program interrupted by ↑C, the user types:

```
JRST 2, @ JOBOPC$X
```

to restore the PC and flags.

D.2 LOADING AND SAVING DDT

Load and save DDT.SAV in 2K of core in the following manner:

Instructions	Example[1]

(1) Load DDT.	`.R LINK`
	`*DSK:DDT/G`
	`LOADER`
	`EXIT`
	`↑C`
	`.ST 149`

(2) ENTER DDT.	`$$H JOBSYM/ =112,,5666`
(3) Type out, in halfword mode, the contents of .JBSYM.	
(4) Open register 6, and put (.JBSYM) (RH) into left half of 6; put ((.JBSYM) (RH) AND[2] 1777) + 2000 into right half of 6.	`6¡ 5666,,3666`
(5) Perform a block transfer stopping at 3777.	`BLT 6,3777$X`
(6) Open .JBSYM; leave the left half as is and change the right half to ((.JBSYM) (RH) AND[2] 1777) + 2000.	`JOBSYM/ =112,,5666 =112,,3666`
(7) Zero memory, except for DDT.	`$$Z`
(8) Start over at 140 to initialize the new symbol table.	`140$G`

[1] ALTMODE is indicated by $.

[2] Logical AND.

(9) Open .JBSA and put DDTEND in the left half and DDT in the right half	`JOBSA! DDTEND,,DDT`
(10) Change back to symbol type-out mode.	`$$S`
(11) Return to monitor.	`^C`
(12) Reduce core to 2K.	`.CORE 2`
(13) Reenter DDT.	`.DDT`
(14) Check that .JBREL is 2K.	`JOBREL/ 3777`
(15) Return to monitor.	`^C`
(16) Save DDT.	`.SAVE DSK DDT`
(17) Check start address.	`.START` `./ 3777`

D.3 EXPLANATION

The DDT-saved file must be saved in 2K (minimum amount of core needed). Also, a starting address must be set up for DDT as location 140. To get DDT into 2K, the DDT symbol table must be moved down to the upper end of the first 2K of core. Any unused locations in DDT should be set to zero ($$Z) and JOBSYM should be set to the new location of the start of the DDT symbol table. Before saving the resulting file, a CORE 2 request should be given to the monitor to ensure that DDT is saved as a 2K core image.

INDEX

APPENDIX E

FORDDT

FORDDT is an interactive program used to debug FORTRAN programs and control their execution. By using the symbols created by the FORTRAN compiler, FORDDT allows you to examine and modify the data and FORMAT statements in your program, set breakpoints at any executable statement or routine, trace your program statement-by-statement, and make use of many other debugging techniques described in this appendix.

Table E-1 lists all the commands available to the user of FORDDT.

Table E-1
Table of Commands

Command	Purpose
Data Access Commands	
ACCEPT	Modifies data locations.
TYPE	Displays data locations.
Declarative Commands	
GROUP	Defines indirect lists for TYPE statements.
MODE	Specifies format of typeout.
OPEN	Accesses program unit symbol table.
PAUSE	Places pause requests.
REMOVE	Removes pause requests.
DIMENSION	Defines dimensions of arrays for FORDDT references. (Unnecessary if /DEBUG:DIMENSIONS was used. See Table B-2.)
DOUBLE	Defines dimensions of double-precision arrays for FORDDT references. (Unnecessary if /DEBUG: DIMENSIONS was used. See Table B-2.)

Table E-1 (Cont.)
Table of Commands

Command	Purpose
Control Commands	
START	Begins execution of FORTRAN program.
CONTINUE	Continues execution after a pause.
GOTO	Transfers control to some program statement within the open program unit.
NEXT	Traces execution of the program.
STOP	Terminates program and returns to monitor mode.
DDT	Enters DDT (if DDT is loaded).
Other Commands	
LOCATE	Lists program unit names in which a given symbol is defined.
STRACE	Displays routine backtrace of current program status.
WHAT	Displays current DIMENSION, GROUP, and PAUSE information.

E.1 INPUT FORMAT

FORDDT commands are made up of alphabetic FORTRAN-like identifiers and need consist of only those characters required to make the command unique. If you wish to specify parameters, a space or tab is required following the command name. FORDDT expects a parameter if a delimiter (i.e., space or tab) is found. Comments may be appended to command lines by preceding the comment with an !.

E.1.1 Variables and Arrays

FORDDT allows you to access and modify the data locations in your program by using standard FORTRAN-10 symbolic names. Variables are specified simply by name. Array elements are specified in the following format:

```
name (S1,...,Sn)
```

where

```
name        = a FORTRAN variable or array name
(S1,...,Sn) = the subscripts of the particular array.
```

You may reference an entire array simply by its unsubscripted name; you may specify a range of array elements by inputting the first and last array elements of the desired range, separated by a dash(-).

Examples

```
ALPHA
ALPHA(7)
ALPHA(PI)
ALPHA(2)-ALPHA(5)
```

E.1.2 Numeric Conventions

FORDDT accepts optionally signed numeric data in the standard FORTRAN-10 input formats:

1. INTEGER - A string of decimal digits.
2. FLOATING-POINT - A string of decimal digits optionally including a decimal point. Standard engineering and double-precision exponent formats are also accepted.
3. OCTAL - A string of octal digits optionally preceded by a double quote (").
4. COMPLEX - An ordered pair of integer or real constants separated by a comma and enclosed in parentheses.

E.1.3 Statement Labels and Source Line Numbers

FORTRAN statement labels are input and output by straightforward numeric reference, i.e., 1234. However, source line numbers must be input to FORDDT with a number sign (#) preceding them. This mandatory sign distinguishes statement labels from source line numbers.

E.2 NEW USER TUTORIAL

The new FORDDT user can rely on the commands described below as a basis for debugging FORTRAN programs. These commands are easy to understand and apply.

E.2.1 Basic Commands

The easiest method of loading and starting FORDDT is:

```
.DEBUG filename.ext (DEBUG)/F10
```

FORDDT will respond with

```
ENTERING FORDDT
>>
```

Just as an asterisk (*) signifies FORTRAN-10's readiness, the two angle brackets signify that FORDDT is awaiting one of the following commands:

OPEN — Makes available to FORDDT the symbol names in a particular program unit of the FORTRAN program. When a program unit symbol table is opened, the previously

open program unit is automatically closed. When FORDDT is entered, the MAIN program is automatically opened. The command format is:

```
OPEN name
```

This will open the particular program unit named and allow all variables within that subprogram to be accessible to FORDDT.

```
OPEN
```

with no arguments will reopen the symbol table of the main program unit.

START Starts your program at the main program entry point. The command format is:

```
START
```

STOP Terminates program execution, causes all files to be closed, and exits to the monitor. The command format is:

```
STOP
```

MODE Defines the display format for succeeding FORDDT TYPE commands. You need type only the first character of the mode to identify it to FORDDT. The modes are:

Mode	Meaning
A	ASCII (left-justified)
C	COMPLEX
D	DOUBLE-PRECISION
F	FLOATING-POINT
I	INTEGER
O	OCTAL
R	RASCII (right-justified)

Unless the MODE command is given, the default typeout mode is the floating-point format.

The command format is:

```
MODE list
```

where list contains one or more of the mode identifiers separated by commas. The current setting can be changed by issuing another MODE command. If more than one mode is given, the values are typed out in the order: F,D,C,I,O,A,R

```
MODE
```

with no arguments will reset FORDDT to the original setting of floating-point format.

TYPE Allows you to display the contents of one or more data locations. They are displayed on your terminal formatted according to the last MODE specification. The command format is:

```
TYPE list
```

where list may contain one or more arrays, variables, array elements, or array element ranges separated by commas. For example:

```
TYPE I, ALPHA, BETA(2),J(3)-J(5)
```

Each item will be displayed in each of the currently active typeout modes as set by the last MODE command.

ACCEPT Allows you to change the contents of a FORTRAN variable, array, array element, or array element range. The command format is:

```
ACCEPT name/mode value
```

where

name = the name of the variable, array, array element, or array element range to be modified. If the field contains an unsubscripted array name or an element range, it causes all the elements to be set to the given value (see special case for ASCII in Section F.6).

mode = the format of the data value to be entered. If given, it must be preceded by a slash (/) and immediately follow the name. (Note that /mode does not apply to FORMAT modification.)

value = the new value to be assigned. It must correspond in format to the given mode.

Data Modes

You need type only the first character of a data mode to identify it to FORDDT. If not specified, the default mode is REAL. The following input modes are available:

Mode	Meaning	Example
A	ASCII(left-justified)	/FOO/
C	COMPLEX	(1.25,-78.E+9)
D	DOUBLE-PRECISION	123.4567890
F	REAL	123.45678
I	INTEGER	1234567890
O	OCTAL	76543210
R	RASCII(right-justified)	\BAR\
S	SYMBOLIC	PSI(2,4)

An example of the ACCEPT command format is:

```
ACCEPT ALPHA 100.6
```

This changes the value of the variable ALPHA to 100.6 with the default input mode of REAL, since mode was not specified.

PAUSE Allows you to set a breakpoint at any label, line number, or subroutine entry in your program. You may set up to ten pauses at one time. When one of these pauses is encountered, execution of the FORTRAN program

is suspended and control is transferred to FORDDT. Also, when a pause is encountered, the symbol table of that subprogram is automatically opened. The command format is:

```
PAUSE P
```

where P is a statement label number, line number, or routine entry point name; for example,

```
PAUSE 100
```

will cause a breakpoint at statement label 100 of the currently open program unit.

Note that subprogram parameter values will be displayed when a pause is encountered at a subprogram entry point.

CONTINUE Allows the program to resume execution after a FORDDT pause. After a CONTINUE is executed, the program either runs to completion, or it runs until another pause is encountered. If you include a value with this command, the program will run until the nth occurrence of the given pause or until a different pause is encountered. The command formats are:

```
CONTINUE
   or
CONTINUE n
```

Example

```
CONTINUE 15
```

will continue execution until the fifteenth occurrence of the pause.

REMOVE Used to remove those pauses from the program previously set up by the PAUSE command. The command format is

```
REMOVE P
```

where P is the number of the statement label where the pause was set, i.e.,

```
REMOVE 100
```

will remove the pause at statement label 100.

Note that REMOVE with no arguments will remove all pauses; therefore, no abbreviation of the command is allowed in this instance. This precaution prevents the accidental removal of all pauses.

WHAT Displays on your terminal the name of the currently open program unit and any currently active pause settings. The command format is:

```
WHAT
```

E.3 FORDDT AND THE FORTRAN-10/DEBUG SWITCH

Most facilities of FORDDT are available without the FORTRAN-10 /DEBUG features; however, if you do not use the /DEBUG switch when compiling a FORTRAN program, the trace features (NEXT command) will not be available, and several of the other commands will be restricted.

Using the /DEBUG switch tells FORTRAN-10 to compile extra information for FORDDT. (See Appendix B, Using the Compiler, for a complete description of each feature.) The additional features include:

1. /DEBUG:DIMENSIONS, which will generate dimension information to the REL file for all arrays dimensioned in the subprogram. The dimension information will automatically be available to FORDDT if you wish to reference an array in a TYPE or ACCEPT command. This feature eliminates the need to specify dimension information for FORDDT by using the DIMENSION command.

2. /DEBUG:LABELS, which will generate labels for every executable source line in the form "line-number L". If these labels are generated, they may be used as arguments with the FORDDT commands PAUSE and GOTO.

   This switch will also generate labels at the last location allocated for a FORMAT statement so that FORDDT can detect the end of the statement. These labels have the form "format-label F". If they are generated, you will be able to display and modify FORMAT statements via the TYPE and ACCEPT commands.

   Note that the :LABELS switch is automatically activated with the :TRACE switch, since labels are needed to accomplish the trace features.

3. /DEBUG:TRACE, which will generate a reference to FORDDT before each executable statement. This switch is required for the trace command NEXT to function.

   Note that if more than one FORTRAN statement has been placed on a single input line, only the first statement will have a FORDDT reference and line-number label associated with it. This also applies to the :LABELS switch.

4. /DEBUG:INDEX, which will force the compiler to store in its respective data location as well as a register the index variable of all DO loops at the beginning of each loop iteration. You will then be able to examine DO loops by using FORDDT. If you modify a DO loop index using FORDDT, it will not affect the number of loop iterations because a separate loop count is used. (See Section D.1.5.)

   Note that this switch has no direct affect on any of the commands in FORDDT.

E.4 LOADING AND STARTING FORDDT

1. The simplest method of loading and starting FORDDT is with the following command string:

```
.DEBUG filename.ext(DEBUG)/F10
```

FORDDT responds with

```
ENTERING FORDDT
>>
```

The angle brackets indicate that FORDDT is ready to receive a command, just as an asterisk (*) signifies FORTRAN-10's readiness.

The DEBUG command to the monitor will also load DDT (standard system debugging program). DDT can be used or ignored, but it does require an extra 2K (octal) of core.

2. You may wish to load your compiled program and FORDDT directly with the LINK-10 loader. (Loading with LINK-10 was accomplished implicitly in the previous command string.) The command sequence is as follows:

```
.R LINK
*filename.ext /DEB/G                          (loads DDT)
*filename.ext /DEB:  FORDDT  /G               (loads FORDDT)
                     FORTRA

*filename.ext /DEB:(DDT,  FORDDT  )/G     loads both DDT
                          FORTRA           and FORDDT
```

If the total FORTRAN program consists of many subroutines and insufficient core is available to complete loading with symbols, it is possible to load with symbols just those sections expected to give trouble. The remaining routines need not be loaded.

E.5 SCOPE OF NAME AND LABEL REFERENCES

Each program unit has its own symbol table. When you initially enter FORDDT, you automatically open the symbol table of the main program. All references to names or labels via FORDDT must be made with respect to the currently open symbol table. If you have given the main program a name other than MAIN by using the PROGRAM statement (see Chapter 5, Section 5.2), FORDDT will ask for the defined program name. After you enter the program name, FORDDT will open the appropriate symbol table. At this point, symbol tables in programs other than the main program can be opened by using the OPEN command. (See Section F.5.)

References to statement labels, line numbers, FORMAT statements, variables, and arrays must have labels that are defined in the currently open symbol table. However, FORDDT will accept variable and array references outside the currently open symbol table, providing the name is unique with respect to all program units in the given load module.

E.6 FORDDT COMMANDS

This section gives a detailed description of all commands in FORDDT. The commands are given in alphabetical order.

ACCEPT Allows you to change the contents of a FORTRAN variable, array, array element, array element range, or FORMAT statement. The command format is:

ACCEPT name/mode value

where

name = the variable array, array element, array element range, or FORMAT statement to be modified.

mode = the format of the data value to be entered. The mode keyword must be preceded by a slash (/) and immediately follow the name. Intervening blanks are not allowed. (Note that /mode does not apply to FORMAT modification.)

value = the new value to be assigned. The format of the input value must correspond to the specified mode.

DATA LOCATION MODIFICATION

Data Modes

The following data modes are accepted:

Mode	Meaning	Example
A	ASCII (left-justified)	/FOO/
C	COMPLEX	(1.25,-78.E+9)
D	DOUBLE-PRECISION	123.4567890
F	REAL	123.45678
I	INTEGER	1234567890
O	OCTAL	76543210
R	RASCII (right-justified)	\BAR\
S	SYMBOLIC	PSI(2,4)

If not specified, the default mode is REAL.

Two-Word Values

For the data modes ASCII, RASCII, OCTAL, and SYMBOLIC, FORDDT will accept a "/LONG" modifier on the mode switch. This modifier indicates that the variable and the value are to be interpreted as two words long.

Example

```
ACCEPT VAR/RASCII/LONG '1234567890'
```

will assume that VAR is two words long and store the given 10-character literal into it.

Initialization of Arrays

If the name field of an ACCEPT contains an unsubscripted array name or a range of array elements, all elements of the array or the specified range will be set to the given value.

Example

```
ACCEPT ARRAY/F 1.0
     or
ACCEPT ARRAY(5)-ARRAY(10)/F 1.0
```

Note that this applies only to modes other than ASCII and RASCII.

Long Literals

When the value field of an ACCEPT contains an unsubscripted array name or range of array elements, and the specified data mode is ASCII or RASCII, the value field is expected to contain a long literal string. ACCEPT will store the string linearly into the array or array range. If the array is not filled, the remainder of the array or range will be set to zero. If the literal is too long the remaining characters will be ignored.

Example

```
ACCEPT ARRAY/RASCII 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

FORMAT STATEMENT MODIFICATION

When the name field of an ACCEPT contains a label, FORDDT expects this label to be a FORMAT statement label and that the value field contains a new FORMAT specification.

Example

```
ACCEPT 10 (1H0,F10.2,3(I2))
```

The new specification cannot be longer than the space originally allocated to the FORMAT by the compiler. The remainder of the area is cleared if the new specification is shorter.

Note that FOROTS performs some encoding of FORMAT statements when it processes them for the first time. If any I/O statement referencing the given FORMAT has been executed, the FORTRAN program has to be restarted (re-initializing FOROTS).

CONTINUE Allows the program to resume execution after a FORDDT pause. After a CONTINUE is executed, the program either runs to completion or until another pause is encountered. The command format is:

```
CONTINUE n
```

where the n is optional and, if omitted, will be assumed to be one. If a value is provided, it may be a numeric constant or program variable, but it will be treated as an integer. When the value n is specified, the program will continue execution until the nth occurrence of this pause. For example,

```
CONTINUE 20
```

will continue execution after the 20th occurrence of the pause.

DDT Transfers control of the program to DDT, the standard system debugging program (if loaded). Any files currently opened by FOROTS are unaffected and return to FORDDT is possible so that program execution may be resumed.

.F10 is the global symbol used to return control to FORDDT. The command format is:

```
.F10$G
```

where $ represents altmode or escape. Your program will be in the same condition as before unless you have modified your core image with DDT.

DIMENSION Sets the user-defined dimensions of an array for FORDDT access purposes. These dimensions need not agree with those declared to the compiler in the source code. FORDDT will allow you to redimension an array to have a larger scope than that of the source program. If this is done, a warning is given. The command format is:

```
DIMENSION S
```

where S is the name of the array specified.

For example:

```
DIMENSION ALPHA(7,5/6,10)
```

FORDDT will remember the dimensions of the array until it is redefined or removed.

The command

```
DIMENSION
```

will give a full list of all the user-defined dimensions for all arrays.

```
DIMENSION ALPHA
```

will display the current information for the array ALPHA only.

```
DIMENSION ALPHA/REMOVE
```

will remove any user defined array information for the array ALPHA.

Arrays, Array Elements, and Ranges

Array elements are specified in the following format:

```
name [d1/d2,...](S1,...)
```

where

name = the name of the array

[...] = optional, and contains dimension information. This form is equivalent in effect to the DIMENSION statement.

(...) = the subscripts of the specific element desired.

The entire array is referenced simply by its unsubscripted name. A range of array elements is specified by inputting the first and last array elements of the desired range separated by a dash (-) (A(5)-A(10)).

DOUBLE Defines the dimensions of a double-precision array. The result of this command is the same as for the DIMENSION command except that the array so dimensioned is understood by FORDDT to be an array with word entries and, therefore, reserves twice the space. The command format is:

DOUBLE arrayname

GOTO Allows you to continue your program from a point other than the one at which it last paused. The GOTO allows you to continue at a statement label or code-generating source line number provided that the /DEBUG:LABELS switch has been used or the contents of a symbol previously ASSIGNed during the program execution.

Note that the program must be STARTed before this command can be used, and also note that a GOTO is not allowed after the ^C^C REENTER sequence. (See F.6.)

The command format is:

GOTO n

GROUP Sets up a string of text for input to a TYPE command. You can store TYPE statements as a list of variables identified by the numbers 1 through 8. This feature eliminates the need to retype the same list of variables each time you wish to examine the same group. Refer to the TYPE command for the proper format of the list.

The command format is:

GROUP n list

where

n = the group number 1-8

list = a string of TYPE statements to be called in future accessing of the current group number.

GROUP

with no arguments will cause FORDDT to type out the current contents of all the groups

GROUP n

will type out the contents of the particular group requested.

Note that one group may call another.

LOCATE Lists the program unit names in which a given symbol is defined. This is useful when the variable you wish to locate is not in the currently open program unit and is defined in more than one program unit. The command format is:

```
LOCATE n
```

where n may be any FORTRAN variable, array, label, line number, or FORMAT statement number.

MODE Defines the default formats of typeout from FORDDT. In initial default mode, variables will be typed in floating-point format. If you wish to change the typeout modes, the command format is:

```
MODE list
```

where list contains one or more of the modes in the following table. (Only the first character of each mode need be typed to identify it to FORDDT.)

Mode	Meaning
F	FLOATING-POINT
D	DOUBLE-PRECISION
C	COMPLEX
I	INTEGER
O	OCTAL
A	ASCII (left-justified)
R	RASCII (right-justified)

A typical command string might be:

```
MODE A,I,OCTAL
```

NEXT Allows you to cause FORDDT to trace source lines, statement labels, and entry point names during execution of your program. This command will only provide trace facilities if the program was compiled with the FORTRAN-10 /DEBUG switch. If this switch was not used, the NEXT command will act as a CONTINUE command. The command format is:

```
NEXT n/sw
```

where

n = a program variable or integer numeric value and

sw = one of the following switches

```
/S= statement label
/L= source line
/E= entry point
```

The default starting value of n is 1, a single statement trace. The default switch is /L.

The command

```
NEXT 20/L
```

will trace the execution of the next 20 source line numbers or until another pause is encountered.

Note that if no argument is specified, the last argument given will be used. For example,

```
NEXT /E
```

will change the tracing mode to trace only subprogram entries using the numeric argument previously supplied.

OPEN — Allows you to open a particular program unit of the loaded program so that the variables will be accessible to FORDDT. Any previously opened program unit is closed automatically when a new one is opened. Only global symbols, symbols in the currently open unit, and unique locals are available at any one time. Note that starting FORDDT automatically opens the MAIN program. The command format is:

```
OPEN name
```

where name is the subprogram name. OPEN with no arguments will reopen the MAIN program.

If the PROGRAM statement was used in the FORTRAN program, the name supplied by you will be requested upon entering FORDDT.

PAUSE — Allows you to place a pause request at a statement number, source line number, or subroutine entry point. Up to ten pauses may be set at any one time. When a pause is encountered, execution is suspended at that point and control is returned to FORDDT. Also, when a pause is encountered, the symbol table of that subprogram is automatically opened.

The command formats include:

```
PAUSE P
PAUSE P AFTER n
PAUSE P IF condition
PAUSE P TYPING /g
PAUSE P AFTER n TYPING /g
PAUSE P IF condition TYPING /g
```

where

P = the point where the pause is requested,
n = an integer constant or variable or array element
g = a group number

```
PAUSE 100
```

will set a pause at statement label 100, cause execution to be suspended, and cause FORDDT to be entered on reaching 100 in the program.

```
PAUSE #245 AFTER MAX(5)
```

will cause a pause to occur at source line number 245 after encountering this point the number of times specified by MAX(5). Note that AFTER may not be abbreviated.

```
PAUSE DELTA IF LIMIT(3,1).GT.2.5E-3
```

If the variable LIMIT(3,1) is greater than the value 2.5E-3, the pause request will be granted. The IF may not be abbreviated, but all the usual FORTRAN logical connectives are allowed.

```
PAUSE 505 TYPING /5
```

will request a pause to be made at the first occurrence of the label 505, and the variables in group 5 will be displayed. The TYPING specification may not be abbreviated.

```
PAUSE LINE#24 AFTER 16 TYPING 3
```

will place a request at source line number 24 after 16 (octal) times through; however, the contents of group 3 will be displayed every time.

When the TYPING option is used with the PAUSE command, control can be transferred to FORDDT at the next typeout by typing any character on the terminal.

Note that pause requests remain after a control C REENTER sequence, a START command, or a control C START sequence.

REMOVE Removes the previously requested pauses. The command format is:

```
REMOVE P
```

For example,

```
REMOVE L#123
```

will remove a pause at program source line number 123.

```
REMOVE ALPHA
```

will remove a pause at the subroutine entry to ALPHA.

REMOVE with no arguments will remove all your pause requests, and, in this case, no abbreviation of REMOVE is allowed. This prevents the unintentional removal of pauses.

START Starts your program at the normal FORTRAN main program entry point. The command format is:

```
START
```

STOP Terminates the program, requests FOROTS to close all open files, and causes an exit to the monitor. The usual command format is:

```
STOP
```

```
STOP/RETURN
```

will allow a return to monitor mode without releasing devices or closing files so that a CONTINUE can be issued.

STRACE Displays a subprogram level backtrace of the current state of the program. The command format is:

```
STRACE
```

TYPE Causes one or more FORTRAN defined variables, arrays, or array elements to be displayed on your terminal. The command format is:

```
TYPE list
```

where list may be one or more variable or array references and/or group numbers. These specifications must be separated by commas, and group numbers must be preceded by a slash (/). The command with no arguments will use the last argument list submitted to FORDDT.

An array element range can also be specified. For example:

```
TYPE PI(5)-PI(13)
```

will display the values from PI(5) to PI(13) inclusive. If an unsubscripted array name is specified, the entire array will be typed.

There are several methods of choosing the form of typeout in conjunction with the MODE command.

1. If you do not specify a format, the defaultis floating-point form.

2. You can specify a format via the MODE command described in this appendix.

3. You can change the format previously designated by the MODE command by including print modifiers in the TYPE or GROUP string. The print modifiers are:

   ```
   /A,/C,/D,/F,/I,/O,/R
   ```

   The first print modifier specified in a string of variables determines the mode for the entire string unless another mode is placed directly to the right of a particular variable. For example, in

   ```
   TYPE /IK,L/O,M,N/A,/2
   ```

   the typeout mode is integer until another mode is specified. Therefore,

   ```
   K,M,and/2 = Integer
   L = OCTAL
   N = ASCII
   ```

WHAT Displays the information saved by FORDDT. The command format is:

```
WHAT
```

E.7 ENVIRONMENT CONTROL

If a program enters an indefinite loop, you can recover by typing a ^C^C REENTER sequence. This action will cause FORDDT to simulate a pause at the point of reentry and allow you to control your run-away program.

Most commands can be used once the program has been reentered; however, GOTO, STRACE, TYPE, and ACCEPT cause transfer of control to routines external to FORDDT. No guarantee can be made to ensure that any of these commands following a ^C^C REENTER sequence will not destroy the user profile. The program must be returned to a stable state before any of these four commands can be issued. In order to restore program integrity, you should set a pause at the next label and then CONTINUE to it. If the /DEBUG:TRACE switch was used, a NEXT 1 command can be issued to restore program integrity.

E.8 FORTRAN-10/OPTIMIZE SWITCH

You should never attempt to use FORDDT with a program that has been compiled with the /OPTIMIZE switch. The global optimizer causes variables to be kept in ACs. For this reason, attempts to examine or modify variables in optimized programs will not work. Also, since the optimizer moves statements around in your program, attempts to trace program flow will lead to great confusion.

E.9 FORDDT MESSAGES

FORDDT responds with two levels of messages - fatal error and warning. Fatal error messages indicate that the processing of a given command has been terminated. Warning messages provide helpful information. The format of these messages is:

```
?FDTXXX text
    or
%FDTXXX text
```

where

```
?    = fatal
%    = warning
FDT  = FORDDT mnemonic
XXX  = 3-letter mnemonic for error message
text = explanation of error
```

Square brackets ([ ]) in this section signify variables and are not output on the terminal.

Fatal Errors

The fatal errors in the following list are each preceded by ?FDT on the user terminal and on listings. They are listed in alphabetical order.

BDF [symbol] IS UNDEFINED OR IS MULTIPLY DEFINED

BOI BAD OCTAL OUTPUT

An illegal character was detected in an octal input value.

CCN CANNOT CONTINUE

Pause has been placed on some form of skip instruction causing FORDDT to loop; should never be encountered in FORTRAN-10 compiled programs.

CFO CORE FILE OVERFLOW

The storage area for GROUP text has been exhausted.

CNU THE COMMAND [name] IS NOT UNIQUE

More letters of the command are required to distinguish it from the other commands.

CSH CANNOT START HERE

The specified entry point is not an acceptable FORTRAN-10 main program entry point.

DTO DIMENSION TABLE OVERFLOW

FORDDT does not have the space to record any more array dimensions until some are removed.

FCX FORMAT CAPACITY EXCEEDED

An attempt was made to specify a FORMAT statement requiring more space than was originally allocated by FORTRAN-10.

FNI FORMAL NOT INITIALIZED

Reference to a FORMAL parameter of some subprogram that was never executed.

FNR [array name] IS A FORMAL AND MAY NOT BE RE-DEFINED

FORMAL parameters may not be DIMENSIONed.

IAF ILLEGAL ARGUMENT FORMAT

The parameters to the given command were not specified properly. Refer to the documentation for correct format.

IAT ILLEGAL ARGUMENT TYPE = [number]

An unrecognized subprogram argument type was detected. Submit an SPR if this message occurs.

ICC COMPARE TWO CONSTANTS IS NOT ALLOWED

Conditional test involves two constants.

IER E (number)

Internal FORDDT error - please report via an SPR.

IGN INVALID GROUP NUMBER

Group numbers must be integral and in the range 1 through 8.

INV INVALID VALUE

A syntax error was detected in the numeric parameter.

ITM ILLEGAL TYPE MODIFIER - S

The mode S is only valid for ACCEPT statements.

LGU [array name] LOWER SUBSCRIPT.GE.UPPER

The lower bound of any given dimension must be less than or equal to the upper bound.

LNF [label] IS NOT A FORMAT STATEMENT

MLD [array name] MULTI-LEVEL ARRAY DEFINITION NOT ALLOWED

The same array cannot be dimensioned more than once (via the [dimensions] construct) in a single command.

MSN MORE SUBSCRIPTS NEEDED

The array is defined to have more dimensions than were specified in the given reference.

NAL NOT ALLOWED

An attempt has been made to modify something other than data or a FORMAT.

NAR NOT AFTER A RE-ENTER

The given command is not allowed until program integrity has been restored via a CONTINUE or NEXT command.

NDT DDT NOT LOADED

NFS CANNOT FIND FORTRAN START ADDRESS FOR [program name]

Main program symbols are not loaded.

NFV [symbol] IS NOT A FORTRAN VARIABLE

Names must be 6-character alphanumeric strings beginning with a letter.

NGF CANNOT GOTO A FORMAT STATEMENT

NPH CANNOT INSERT A PAUSE HERE

An attempt has been made to place a pause at other than an executable statement or subprogram entry point.

NSP [symbol] NO SUCH PAUSE

An attempt has been made to REMOVE a pause that was never set up.

NUD [symbol] NOT A USER DEFINED ARRAY

An attempt has been made to remove dimension information for an array that was never defined.

PAR PARENTHESES REQUIRED (..)

Parentheses are required for the specification of FORMAT statements and complex constants.

PRO TOO MANY PAUSE REQUESTS

The PAUSE table has been exhausted. The maximum limit is 10.

SER SUBSCRIPT ERROR

The subscript specified is outside the range of its defined dimensions.

STL [array name] SIZE TOO LARGE

An attempt has been made to define an array larger than 256K.

TMS TOO MANY SUBSCRIPTS

The array is defined to have fewer dimensions than are specified in the given element reference.

URC UNRECOGNIZED COMMAND

Warning Messages

Each warning message in this list is preceded by %FTN on your terminal and on listings. They are given here in alphabetical order.

ABX [array name] COMPILED ARRAY BOUNDS EXCEEDED

FORDDT has detected another symbol defined in the specified range of the array. Note that this will occur in certain EQUIVALENCE cases and can be ignored at that time.

CHI CHARACTERS IGNORED: "[text]"

The portion of the command string included in "text" was thought to be extraneous and was ignored.

NAR [symbol] IS NOT AN ARRAY

NSL NO SYMBOLS LOADED

FORDDT cannot find the symbol table.

NST NOT STARTED

The specified command requires that a START be previously issued to ensure that the program is properly initialized.

POV PROGRAM OVERLAYED

The symbol table is different from the last time FORDDT had control.

SFA SUPERSEDES F10 ARRAY

The FORTRAN-10 generated dimension is being superseded for the given array.

SPO VARIABLE IS SINGLE-PRECISION ONLY

XPA ATTEMPT TO EXCEED PROGRAM AREA WITH [symbol name]

An attempt has been made to access memory outside the currently defined program space.

decsystem10

FILCOM

File Comparison Program

digital equipment corporation · maynard. massachusetts

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this manual.

The software described in this document is furnished to the purchaser under a license for use on a single computer system and can be copied (with inclusion of DIGITAL's copyright notice) only for use in such system, except as may otherwise be provided in writing by DIGITAL.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

The HOW TO OBTAIN SOFTWARE INFORMATION page, located at the back of this document, explains the various services available to DIGITAL software users.

The postage prepaid READER'S COMMENT form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

FILCOM PROGRAM

FUNCTION

The FILCOM program is used to compare two versions of a file and to output any differences. Generally, this comparison is line by line for ASCII files or word by word for binary files. FILCOM determines the type of comparison to use by examining either switches specified in the command string or the extension of the files. Switches always take precedence over file extensions.

COMMAND FORMAT

```
.R FILCOM
*output dev:file.ext [directory] = input dev (1):file.ext[directory], input dev (2):
     file.ext [directory]
```

output dev: = the device on which the differences are to be output.

input dev: = the device on which an input file resides.

DEFAULTS

1. If the entire output specification is omitted, the output device is assumed to be TTY. However, the equal sign must be given to separate the input and output specifications of the command string.

2. If an output filename is specified, the default output is DSK.

3. If the output filename is omitted, the second input filename is used, unless it is null. In this case, the filename FILCOM is used.

4. If the output extension is omitted, .SCM is used on a source compare and .BCM is used on a binary compare.

5. If the [directory] is omitted (input or output side), the user's default directory is assumed.

6. If an input device is omitted, it is assumed to be DSK.

7. If the filename and/or extension of the second input file is omitted, it is taken from the first input file.

8. A dot following the filename of the second input is necessary to explicitly indicate a null extension, if the extension of the first input file is not null. For example, to compare FILE.MAC and FILE. (i.e., with null extension), use the following command string:

```
.R FILCOM
* = FILE.MAC,FILE.
```

9. The second input file specification cannot be null unless a binary compare is being performed. In a binary compare, if the first input file is not followed by a comma and a second input file descriptor, the input file is compared to a zero file and is output in its entirety. This gives the user a method of listing a binary file. Refer to Example 4.

SWITCHES

The following switches can appear in the command string, depending on whether a source compare or a binary source compare is being performed.

Binary Compare

/H	Type list of switches available (help text from device SYS:).
/nL	Specify the lower limit for a partial binary compare (n is an octal number). This switch, when used with the /nU switch, allows a binary file to be compared only within the specified limits.
/Q	When the files are different, print the message ?FILES ARE DIFFERENT, but do not list the differences. This switch is useful when BATCH control files want to test for differences but do not want the log file filled with these differences.
/nU	Specify the upper limit for a partial binary compare (n is an octal number). This switch, when used with the /nL switch, allows a binary file to be compared only within the specified limits.
/W	Compare files in binary mode without expanding the files first (refer to Appendix D of the Operating System Commands Manual). This switch is used to compare two binary files with ASCII extensions.
/X	Expand SAV files before comparing them in binary mode. This action removes differences resulting from zero compression (refer to Appendix D of the Operating System Commands Manual).

Source Compare

/A	Compare files in ASCII mode. This switch is used to force a source compare on two ASCII files.

/B Compare blank lines. Without this switch, blank lines are ignored.

/C Ignore comments (all text on a line following a semicolon) and spacing (spaces and tabs). This switch does not cause a line consisting entirely of a comment to become a blank line, which is normally ignored.

/H Type list of switches available (help text from device SYS:).

/nL Specify the number of lines that determine a match (n is an octal number). A match means that n successive lines in each input file have been found identical. When a match is found, all differences occurring before the match and after the previous match are output. In addition, the first line of the current match is output after the differences to aid in locating the place within each file at which the differences occurred. The default value for n is 3.

/Q Print the message ?FILES ARE DIFFERENT, when tne files are different, but do not list the differences.

/S Ignore spaces and tabs.

/U Compare in update mode. This means that the output file consists of the second input file with vertical bars (or back slashes for 64-character printers) next to the lines that differ from the first input file. This feature is useful when updating a document because the changes made to the latest edition are flagged with change bars in the left margin. The latest edition of the document is the second input file.

If switches are not specified in the command string, the files are compared in the mode implied by the extension. The following extensions are recognized as binary and cause a binary compare if one or both of the input files have one of the extensions.

.BAC	.HGH	.RMT
.BIN	.LOW	.RTB
.BUG	.MSB	.SAV
.CAL	.OVR	.SFD
.CHN	.QUE	.SHR
.DAE	.QUF	.SVE
.DCR	.REL	.SYS
.DMP	.RIM	.UFD
		.XPN

Binary files are compared word by word starting at word 0 except for the following two cases:

1. Files with extensions .SHR and .HGH are assumed to be high segment files. Since the word count starts at 400000, upper and lower limits, if used, must be greater than (or equal to in the case of the lower limit) 400000.

2. Files with extension .SAV, .LOW, and .SVE are assumed to be compressed core image files and are expanded before comparing.

Conflicts are resolved by switches or defaults. If a conflict arises in the absence of switches, the files are assumed to be ordinary binary files.

OUTPUT

In most cases, headers consisting of the device, filename, extension, and creation date of each input file are listed before the differences are output. However, headers do not appear on output from the /U switch (update mode on source compare).

Source compare output – After the headers are listed, the following notation appears in the left column of the output

n)m

where

n is the number of the input file, and
m is the page number of the input file (see examples).

The right column lists the differences occurring between matches in the input files. Following the list of differences, a line identical to each file is output for reference purposes.

The output from the /U switch differs from the above-described output in that the output file created is the second input file with vertical bars in the left column next to the lines that are different from the first input file.

Binary compare output – When a difference is encountered between the two input file, a line in the following format appears on the output device:

octal loc. first file-word second file-word XOR of both words

If the exclusive OR (XOR) of the two words differs only in the right half, the third word output is the absolute value of the difference of the two right halves. This usually indicates an address that changed.

If one input file is shorter than the other, after the end of file is encountered on the shorter file, the remainder of the longer file is output.

CHARACTERISTICS

The R FILCOM command:

Places the terminal in user mode.

Runs the FILCOM program, thereby destroying the user's core image.

ASSOCIATED MESSAGES

?2K CORE NEEDED AND NOT AVAILABLE

FILCOM needs 2K of core to initialize I/O devices and this core is not available from the monitor.

?BUFFER CAPACITY EXCEEDED AND NO CORE AVAILABLE

The buffer is not large enough to handle the number of lines required for looking ahead for matches, and additional core is not available.

?COMMAND ERROR

One of the following errors occurred in the last command string typed.

1. There is no separator (← or =) between the output and input specifications.
2. The input specification is completely null.
3. The two input files are not separated by a comma.
4. A file descripter consists of characters other than alphanumeric characters.
5. FILCOM does not recognize the specified switch.
6. The project-programmer number is not standard format, i.e., [proj,prog].
7. The value of the specified switch is not octal.
8. The first input file is followed by a comma but the second input file is null.

?DEVICE dev: NOT AVAILABLE

Device is assigned to another job or does not exist.

?FILE n NOT IN SAV FORMAT

The user indicated via the /X switch that the file is to be expanded but the specified file is not compressed file format. n is either 1 or 2 indicating the first file or the second file.

?FILE n READ ERROR

An error has occurred on either the first or second input device.

%FILES ARE DIFFERENT

The two input files specified in the command string are different (i.e., the two files are not two versions of the same file but are two different files).

?INPUT ERROR – file.ext FILE NOT FOUND

The specified file could not be found on the input device.

NO DIFFERENCES ENCOUNTERED

No differences were found between the two input files.

?OUTPUT DEVICE ERROR

An error has occurred on the output device.

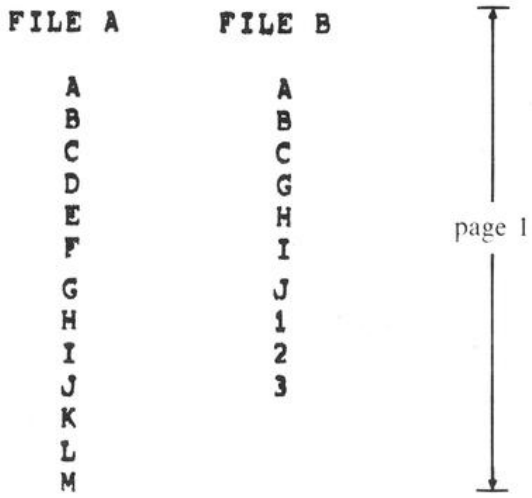
?OUTPUT INITIALIZATION ERROR

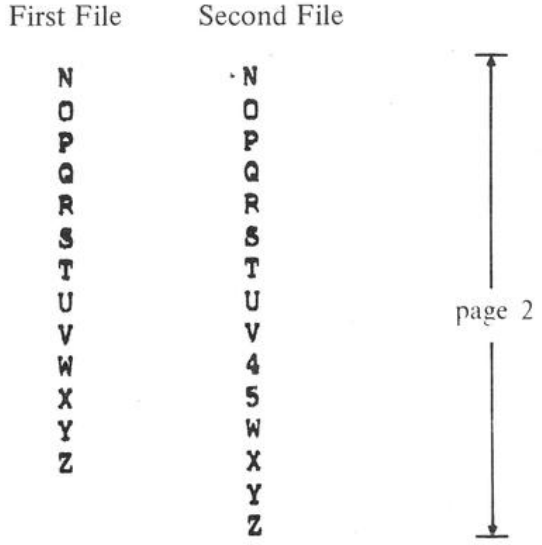
The output device cannot be initialized for one of the following reasons:

1. The device does not exist or is assigned to another job.
2. The device is not an output device.
3. The file cannot be placed on the output device.

Examples

1. The user has the following two ASCII files on disk:

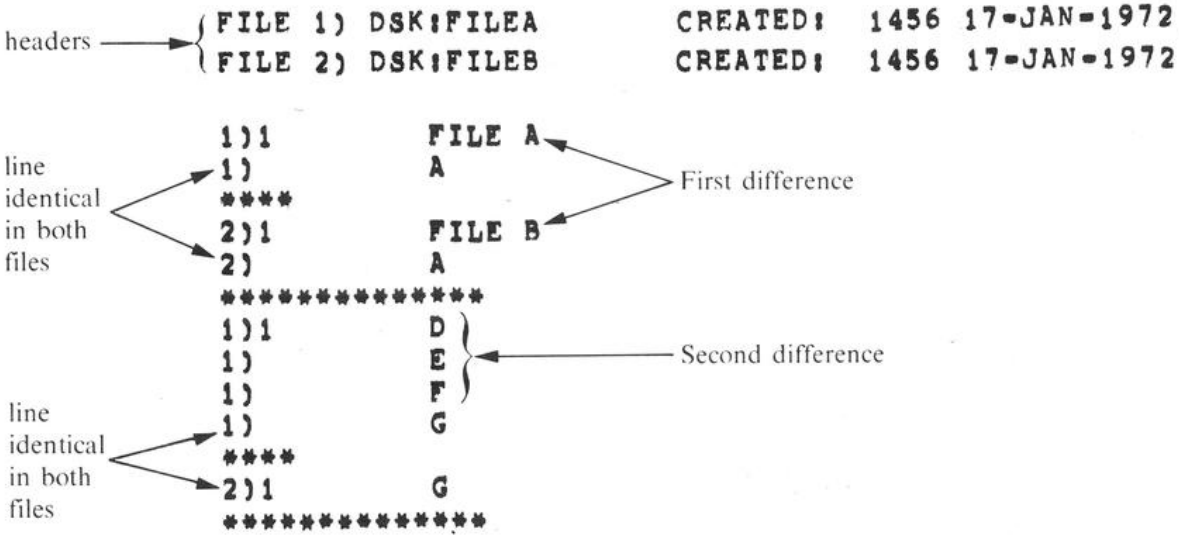


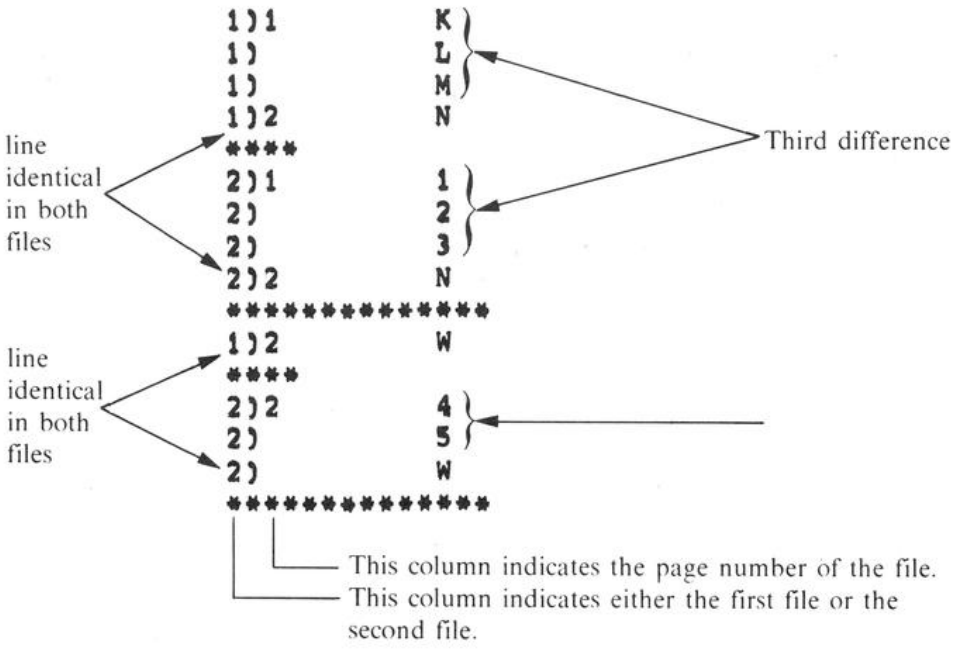


To compare the two files and output differences on the terminal, the following sequence is used:

```
.R FILCOM
```
Run the FILCOM program.

```
* = FILEA,FILEB
```
Compare the two files on disk and output the differences on the terminal. By default, three consecutive identical lines determine a match.





To compare the two files and output the differences on the line printer, the following commands are used. Note that in this example the number of successive lines that determine a match has been set to 4 with the /4L switch.

```
.R FILCOM
*LPT:/4L = FILEA,FILEB

FILE 1) DSK:FILEA      CREATED:  1456 17-JAN-1972
FILE 2) DSK:FILEB      CREATED:  1456 17-JAN-1972

1)1     FILE A
1)      A
1)      B
1)      C
1)      D
1)      E
1)      F
1)      G
****
2)1     FILE B
2)      A
2)      B
2)      C
2)      G
**********
```

These lines are listed as being different because the /4L switch specifies that 4 consecutive line must be found identical in the two files before they are considered as a match.

```
1)1       K
1)        L
1)        M
1)2       N
****
2)1       1
2)        2
2)        3
2)2       N
***************
1)2       W
****
2)2       4
2)        5
2)        W
***************
```

To compare the two files so that the second input file is output with vertical bars in the left column next to the lines that differ from the first input file, use the following command sequence.

```
.R FILCOM↵
*LPT:/U = FILEA,FILEB↵

    |  FILE B

       A
       B
       C
    |  G
       H
       I
       J
    |  1
    |  2
    |  3
       N
       O
       P
       Q
       R
       S
       T
       U
       V
    |  4
    |  5
       W
       X
       Y
       Z
```

The lines with vertical bars indicate the differences between the two files.

2. To compare two binary files on disk and output the differences on the terminal, use the following command sequence.

```
.R FILCOM
*TTY:=DSK:DIAL,REL,DIAL2
FILE 1) DSK:DIAL,REL    CREATED: 0000 23-DEC-1971
FILE 2) DSK:DIAL2,REL   CREATED:  0000 12-AUG-1971

000000  000004 000001  000004 000060          000057
000002  000000 054716  000311 372712  000311 326004
000003  000006 000001  017573 510354  017575 510355
000004  000000 000000  017573 513216  017573 513216
```

3. To compare two high segment file, the command sequence below is used. Note that the location begins at 400000.

```
.R FILCOM
*TTY:=SYS:TABLE.SHR, TABLE.SHR
FILE 1) SYS:TABLE.SHR   CREATED: 2020 24-JAN-1972
FILE 2) DSK:TABLE.SHR   CREATED: 1829 30-NOV-1971

400000  001611 400010  001630 407157  000021 007147
400003  006675 000000  015024 407670  013651 407670
400004  005600 000070  004700 000113  001100 000163
400005  545741 444562  554143 625700  011602 261262
400010  634000 000000  260740 403516  454740 403516
400011  474000 000000  200000 414036  674000 414036
400012  402000 000156  202000 000720  600000 000676
400013  200040 406354  201000 000472  001040 406726
```

4. To list a binary file, use the following command sequence.

```
.R FILCOM
*TTY:=SYS:DOT.REL
000000  000004 000001
000001  000000 000000
000002  000000 054716
000003  000006 000001
000004  000000 000000
000005  000007 517716
000006  000001 0000002
000007  000000 000000
 .
 .
 .
```

5. To compare two binary files between locations 150-160 (octal).

```
.R FILCOM
*TTY:/150/160U=SYS:SYSTAT.SAV,SYS:SYSDPY.SAV
FILE 1) SYS:SYSTAT.SAV  CREATED: 0818 30-NOV-1971
FILE 2) SYS:SYSDPY.SAV  CREATED: 1642 29-NOV-1971

000150  200400 000137  200740 003217 000340 003320
000151  260740 004226  404500 004242 664240 000064
000152  260740 004253  66 002000 401240 006253
000153  200040 005011  260740 002723 060700 007732
000154  260740 004063  200040 004243 060700 000220
000155  201041 777777  202040 003241 003001 774536
000156  047040 000042  200040 004241 247000 004203
000157  254000 000174  251040 004142 005040 004036
000160  476000 006774  211040 000144 667040 006630
```

6. To compare two .SAV files. Note that the files are expanded before the comparison.

```
.R FILCOM
*TTY:=SYS:TRY1.SAV,SYS:TRY.SAV
FILE 1) SYS:TRY1.SAV   CREATED: 2043 05-JAN-1972
FILE 2) SYS:TRY.SAV    CREATED: 0818 30-NOV-1971

000114  004000 000140  000000 000000 004000 000140
000116  777536 005536  000000 000000 777536 005536
000117  000000 005536  000000 000000        005536
000120  006000 000140  007222 000140 001222 000000
000121  000000 006000  000000 007222        001222
000130  010000 000005  000000 000000 010000 000005
000133  003727 005777  006643 007777 005164 002000
000137  003400 000070  046700 000004 045300 000074
000140  264000 001454  047000 000000 223000 001454
000141  260040 001773  200040 005075 060000 004706
000142  201240 001447  402000 006644 603240 007203
000143  542240 001634  251040 007221 713200 006416
000144  260040 002774  403000 000015 663040 002761
000145  621000 000010  476000 006715 257000 006705
000146  200240 003504  200740 006606 000500 005302
000147  251240 000012  051140 005076 200300 005064
000150  402000 003613  200400 000137 602400 003724
000151  201040 003730  260740 004226 061700 007516
000152  200260 003632  260740 004253 060520 007461
000153  321240 000164  200400 005011 121200 005175
```

decsystem10

FILEX

File Transfer Program

digital equipment corporation · maynard. massachusetts

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this manual.

The software described in this document is furnished to the purchaser under a license for use on a single computer system and can be copied (with inclusion of DIGITAL's copyright notice) only for use in such system, except as may otherwise be provided in writing by DIGITAL.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

The HOW TO OBTAIN SOFTWARE INFORMATION page, located at the back of this document, explains the various services available to DIGITAL software users.

The postage prepaid READER'S COMMENT form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

FILEX

1.0 INTRODUCTION

FILEX is a general file transfer program intended to convert between various core image formats, and to read and write various DECtape directory formats as well as standard disk files.

The commands to FILEX are similar to those in a PIP command string. Files are transformed as 36-bit binary data. No processing is done on the data itself except that necessary to convert between various core image representations.

Rapid tape processing via a disk scratch file is available.

"Wild-card" filenames (*) are permitted.

2.0 DEVICE FORMATS AVAILABLE

Non-DECtape devices are read and written in binary. Device, filename, extension, project-programmer number, and protection are supplied in the usual way.

DECtapes in the usual PDP-10 directory format may be read or written in binary in the usual way. They may also be read via a disk scratch file, which is much faster for either a tape with many files, or a tape which has been written by TENDMP (with consecutive blocks allocated to the same file).

Similarly, DECtapes may be read (with or without use of a scratch file) and written in either the old DEC PDP-6 DECtape format, or the MIT project MAC PDP-6/10 DECtape format. For both of these formats, the monitor's DECtape service routine cannot be made to run efficiently, so the scratch file technique is advised. The /O (old) and /M (mac) switches specify these formats. /T (ten) returns to PDP-10 format tapes.

3.0 DATA FORMATS AVAILABLE

Unless one of the following special formats applies, all files are transferred unmodified as 36-bit binary data.

Core image files are the special cases handled. Processing is available to convert from any of the following formats to any other of them. If the input and output formats are identical, the file is simply copied.

Each of the following core image formats is indicated by specific extensions, all of which may be overridden by switches.

3.1 Save-file Format

This format is assumed for files with extensions .SAV, .LOW, and .SVE, and can be forced by the /C switch (compressed core image). The default output extension for a /C file is .SAV.

3.2 Expanded Core Image File (as used by FILDDT)

This format is assumed for files with the extension .XPN, and can be forced by the /E switch (expanded). The default output extension for a /E file is .XPN.

3.3 Dump Format (old PDP-6 version of save)

This format is assumed for files with extension .DMP, and can be forced by the /D switch.

3.4 SBLK Format (simple block)

This is Project MAC's equivalent ot DEC's .SAV format. It is not assumed for any extension, but is forced by the /S switch. The default output extension for a /S file is .BIN.

3.5 Binary Processing

The /B switch causes binary processing even though a file has one of the above special extensions.

4.0 COMMAND FORMAT

A FILEX command is of the form:

```
* output specifier ← input specifier(s)
                or
* output specifier = input specifier(s)

.R FILEX
*dev:ofile.ext[directory]<nnn>/switches=
dev:file.ext[directory]/switches
```

If the project-programmer number and/or a switch appear after a device, they apply to all following files. If they appear after a filename, they apply only to that file.

The input name or extension may be an asterisk (*), in which case the usual wild-card processing occurs.

The output name or extension may be an asterisk (*), in which case the name or extension of the input file is copied.

If the output name or extension is missing, almost the same processing occurs as for an asterisk (*), except that all core image files will be written with the default extension and format appropriate to the output device (unless overridden by switches). That is,

```
*DSK:←DTA1:FOO.DMP/O
```

would cause the DMP format file to be compressed (/C) and written as FOO.SAV. If protection <nnn> is not specified, files are written with the system standard protection unless the files are being written on SYS. On SYS, files are written with <155>, except for files with extension .SYS. These files have the default protection of <157>.

5.0 DECTAPE PROCESSING SWITCHES

To cause an input DECtape to be processed quickly via a scratch file, use the /Q (quick) switch.

To cause the /Q processing and preserve the scratch file after processing for use by another command, use the /P (preserved quick) switch.

To reuse a scratch file preserved by /P in a previous command, use the /R (reuse) switch.

To ignore read errors on the input device, use the /G (go on) switch.

FILEX checks the always-bad-checksum bit in the level D disk format, so /G is not needed for those files with .RPABC on (e.g., CRASH.SAV).

To copy a CRASH.SAV file to an expanded format file for FILDDT to examine, type (for example):

```
DSK:SER106.SAV[10,10]/E←DSKC:CRASH.SAV[1,4]
```

while logged in as [1,2] (to be able to read CRASH.SAV, which is read-protected by the refresher).

The /Z switch on an output file, if it is on a DECtape, causes the appropriate format of the zeroed directory to be written on the tape. If the string

```
↑TAPEID
```

appears in the output specifier, then TAPEID is written as the tape identifier in the directory. TAPEID may be 6 characters on a PD-10 tape, 3 characters on a Project MAC tape, and is not present on a PDP-6 tape.

The /L switch on an input DECtape file causes the tape directory to be typed on the TTY. Do not put TTY: in the output file specifier. That would try to write files on the TTY in binary.

6.0 SUMMARY OF FILEX SWITCHES

Meaning of Switches:

Help text

/H to obtain an explanation of the command string and individual switches.

6.1 DECtape Format Specifiers

/F PDP-15 DECtape format

/M MIT project MAC PDP-6/10 DECtape format

/O Old DEC PDP-6 DECtape format

/T normal PDP-10 directory format

/V PDP-11 DECtape format (Note that PDP-11 contiguous files are not supported by FILEX.)

6.2 File Format Specifiers

/A ASCII processing; meaningful only for PDP-11 and PDP-15 tapes.

/B binary processing; overrides default extension. Files read from a PDP-11 format tape with this switch contain four 8-bit bytes in each 36-bit word (1st byte in bits 10-17, 2nd byte in bits 2-9, 3rd byte in bits 28-35, and 4th byte in bits 20-27). Files written on a PDR-11 format tape with this switch are assumed to have the same format.

/C compressed; save file format. This format is assumed for files with extensions .SAV, .LOW, .SVE. The default output extension is .SAV unless the input extension is .LOW or .SVE, in which case the extension remains unchanged.

/D dump format. This format is assumed for files with extension .DMP.

/E expanded core image files (used by FILDDT). This format is assumed for files with extension .XPN. The default output extension is .XPN.

/I Image processing; meaningful only for PDP-11 and PDP-15 tapes.

/S simple block (SBLK) format, project MAC's equivalent of .SAV format. The default output extension is .BIN.

6.3 DECtape Processing Specifiers

/G (go on), ignores read errors on input device. FILEX checks the always-bad-checksum bit in the 5-series monitor, so this switch is not needed for files with .RPAC on (e.g., CRASH.SAV).

/L (list), causes a directory on an input DECtape file to be typed on the terminal, or causes a directory listing of the output DECtape at the end (i.e., after the output).

/P (preserved), causes quick processing (/Q) and preserves the scratch file after processing for use by another command.

/Q (quick), causes an input or output DECtape to be processed quickly by creating a scratch file on disk. This file is deleted after processing is completed.

/R (reuse), reuses a scratch file preserved by a /P in a previous command.

/Z (zero), causes the appropriate format of a zero directory to be written on a DECtape output file. (Zeroing a DECtape directory is equivalent to deleting all the files on the tape.) If TAPEID appears in the output specifier, then TAPEID is written as the tape identifier in the directory. TAPEID is preceded by an up arrow (↑) and may be 6 characters on a PDP-10 tape, 3 characters on a project MAC tape, and is not present on a PDP-6 tape.

Characteristics

The R FILEX command:

Runs the FILEX program, thereby destroying the user's core image.

Examples

```
.R FILEX
*DSK:=DTA1:TEST.DMP/C
```

The dump format file is compressed and written as TEST.SAV.

Copy CRASH.SAV to an expanded format file for FILDDT to examine.

```
.R FILEX
*DSK:=SER105.XPN[10,1]=DSKC:CRASH.SAV[1,4]
```

decsystem10

GLOB
Global Symbol Listing

digital equipment corporation · maynard. massachusetts

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this manual.

The software described in this document is furnished to the purchaser under a license for use on a single computer system and can be copied (with inclusion of DIGITAL's copyright notice) only for use in such system, except as may otherwise be provided in writing by DIGITAL.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

The HOW TO OBTAIN SOFTWARE INFORMATION page, located at the back of this document, explains the various services available to DIGITAL software users.

The postage prepaid READER'S COMMENT form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

GLOB PROGRAM

FUNCTION

The GLOB program reads multiple binary program files and produces an alphabetical cross-referenced list of all the global symbols (symbols accessible to other programs) encountered. This program also searches files in library search mode, checking for globals, if the program file was loaded by the LOADER in library search mode.

The GLOBAL program has two phases of operation; the first phase is to scan the input files and build an internal symbol table, and the second, to produce output based on the symbol table. Because of these phases, the user can input commands to GLOB in one of two ways. The first way is to specify one command string containing both the output and input specifications. (This is the command string format most system programs accept.) The second is to separate the command string into a series of input commands and output commands.

COMMAND FORMATS

1. R GLOB

   outdev:file.ext[directory]=input dev:file.ext[directory]
   file.ext, . . . ,dev:file.ext[directory]

2. R GLOB

   followed by one or more input commands in the form

   dev:file.ext[directory],file.ext[directory], . . . ,
   dev:file.ext[directory], . . .

   and then one or more output commands in the form

   outdev:file.ext[directory]=

When the user separates his input to GLOB into input commands and output commands. (Command Format #2), the input commands contain only input specifications and the output commands, only output specifications. Each output command causes a listing to be generated; any number of listings can be printed from the symbol table generated from the current input files as long as no input commands occur after the first output command. When an input command is encountered after output has been generated, the current symbol table is destroyed and a new one begun.

DEFAULTS

1. If the device is omitted, it is assumed to be DSK. However, if the entire output specification is omitted, the output device is TTY.

2. If the output filename is omitted, it is the name of the last input file on the line (Command Format #1) or is GLOB if the line contains only output commands (Command Format #2). The input filenames are required.

3. If the output extension is omitted, .GLB is used. If the input extension is omitted, it is assumed to be .REL unless the null extension is explicitly specified by a dot following the filename.

4. If the project-programmer number [ppn] is omitted, the user's default directory is used.

5. An ALTmode terminates the command input and signals GLOB to output the cross-referenced listing. In other words, a listing is not output until GLOB encounters an ALTmode. The ALTmode appears at the end of the command string shown in Command Format #1 or at the end of each output command shown in Command Format #2.

SWITCHES

Switches control the type of global listing to be output. Each switch can be preceded by a slash, or several switches can be enclosed in parentheses. Only the most frequently specified switch (except for L, M, P, Q, and X, which are always in effect) is in effect at any given time. If no switches are specified, all global symbols are output. The following switches are available.

/A Output all global symbols. This is the default if no switches are specified.

/E List only erroneous (multiple defined or undefined) symbols.

/F List nonrelocatable (fixed) symbols only.

/H List the switches available (help text) from SYS:GLOB.HLP.

/L Scan programs only if they contain globals previously defined and not yet satisfied (library search mode).

/M Turn off library search mode scanning resulting from a /L switch.

/N List only symbols which are never referenced.

/P List all routines that define a symbol to have the same value. The routine that defines the symbol first is listed followed by a plus (+) sign. Subsequent routines that define the symbol are listed preceded by a plus sign.

/Q Suppress the listing of subsequent definers that result from the /P switch.

/R List only relocatable symbols.

/S List symbols with non-conflicting values that are defined in more than one program.

/X Do not print listing header when output device is not the terminal, and include listing header when it is the terminal. Without this switch, the header is printed on all devices except the terminal. The listing header is in the following format:

FLAGS SYMBOL OCTAL VALUE DEFINED IN REFERENCED IN

Symbols listed are in alphabetical order according to their ASCII code values. The octal value is followed by a prime (') if the symbol is relocatable. The value is then relative to the beginning of the program in which the symbol is defined. Flags preceding the symbol are shown below.

M Multiply defined symbol (all values are shown).

N Never referred to (i.e., was not declared external in any of the binary programs).

S Multiply specified symbol (i.e., defined in more than one program but with non-conflicting values). The name of the first program in which the symbol was encountered is followed by a plus sign.

U Undefined symbol.

CHARACTERISTICS

The R GLOB command:

Places the terminal in user mode.

Runs the GLOB program, thereby destroying the user's core image.

ASSOCIATED MESSAGES

?COMMAND SYNTAX ERROR
TYPE/H FOR HELP

An illegal command string was entered.

?DESTINATION DEVICE ERROR

An I/O error occurred on the output device.

?ENTER ERROR n
?DIRECTORY FULL

No additional files can be added to the directory of the output device; n is the disk error code.

?ILLEGAL SWITCH

A non-recognizable switch was used in the command string.

?LOOKUP ERROR n
?file.ext FILE NOT FOUND

The named file cannot be found in the directory on the specified device.

?dev NOT AVAILABLE

The requested device does not exist or is assigned to another job.

?TABLE OVERFLOW – CORE UUO FAILED TRYING TO EXPAND TO xxx

The GLOB program requested additional core from the monitor, but none was available.

Examples

```
.R GLOB
*LPT:=MAIN,DTA2:SUB40,SUB50(ESC)

*DTA4:BATCH.REL,DTA,DTA6:NUMBER.REL,CLASS
*DSK:MATH.REL,LIBARY.

*LPT:=/F(ESC)

*DSK:SYMBOL=/R(ESC)
```

```
*TTY:=/E$
U EXTSYM SUBRTE
```

```
*^C
```

decsystem10

OPSER

Operator Service Program

digital equipment corporation · maynard. massachusetts

OPSER

1.0 FUNCTION

OPSER facilitates multiple job control for the operator by running jobs on a sublevel over pseudo-teletypes (mnemonic PTY). Previous to OPSER, when the operator had to run more than one job at one time, he either had to run some detached, or use several terminals. In the former case, no I/O link existed between the running job and the operator, and if he was not watching, it would go by without notice. Using several terminals was problematic since the operator had to run from console to console to service the programs. By running these system jobs over PTY's, OPSER maintains an excellent I/O link, and concentrates in one console the entire control center of the system. It need not run only as an operator job, regular users can employ OPSER to fan out multiple jobs.

2.0 INTERACTIVE COMMANDS

OPSER signifies its readiness to process any of the following commands by typing * or ! at the start of the line. * will be typed if no subjobs are in use, or to indicate subjobs that are in terminal input wait. Thus, when OPSER types * to signal command wait, nothing will happen until the operator acts. The commands must be typed to sufficient length to make them unique. All are prefixed by a colon (:).

↑A (ctrl–A)	Send ↑C(ctrl–C) to the subjob.
:AUTO /hh:mm filename	Process the named file as a list of interactive commands; the AUTO file is terminated by the end-of-file or by the typing of a line on the console by the operator; AUTO files may call other files, including themselves. /hh:mm is optional; if it is included, the auto file will not be run until that time. If that time has passed, the auto file is run immediately.
:AUTO /+hh:mm filename	Process after the amount of time specified by the +hh:mm has elapsed.
:AUTO />hh:mm filename	Process at the next occurrence of hh:mm.
:AUTO /<hh:mm filename	Do not process if time has already past hh:mm.
↑B	Send ↑O(ctrl–O) to the subjob.
↑C	Same as :EXIT.
:CLOSE	Close the log file without opening a new one.

:CONTINUE	Continue processing an auto file that was interrupted by control-C. This allows the operator to gain control of a subjob during auto file processing.
:CURRENT	Type the number of the current subjob (the last one referenced).
:DAYTIME	List today's date and current time.
:DEFINE xx=n	Define xx as the mnemonic for subjob n. (Subjob specification is explained in Section 3.0.)
:DEVICE nam:logn:n	Reassign the device with the physical name nam, and the logical name logn to subjob n. The logical name need not be present, but a null field must be typed, e.g., to reassign the CDR to subjob 3, one must type :DEVICE CDR::3.
:ERROR n,m,p	Ignore all non-error messages from the specified subjobs. An error message is any line beginning with a question mark (?) or percent sign (%). Reset by :REVIVE.
:EXIT	Exit to the monitor if no subjobs are in use; otherwise give a list of those that are running.
:FREE	Type the number of the first unused subjob.
:HELP	Type a text which briefly explains the commands and how to communicate with subjobs.
:JCONT nn	Continue a job waiting for operator action.
:KJOB n,m,p	Kill the specified subjobs saving all files. Causes /Z:0 to be sent to KJOB so spooled files are not queued.
:KILL n,m,p	Same as KJOB.
:KSYS hhmm	Stop timesharing at the time specified by hhmm. If +hhmm is used, it means the number of hours and/or (minimum of 5 minutes if + form is used) minutes from the current time.
:LOGIN p,pn	Send the LOGIN line over the first free subjob; if no project-programmer number is typed, assume OPSER's project-programmer number.

:MONITOR	Return to monitor level.
:MSGLEVEL 0	Add JOBSTS bits to the typed response to the :WHAT command. (default)
:QUEUE <line>	Initiate the first free subjob and send that line to the system queue manager.
:RESOURCES	List the available resources of the system.
:RESTRICT dev1:, dev2:, . . . ,devn:	Restricts the specified device to the operator's use. The operator can assign the device to a user, but the user cannot reassign it to anyone but the operator. (The colon after the device name is optional.)
:REVIVE n	Resume normal echoing of output from subjob n.
:SEND <line>	Simulate the monitor command SEND.
:SET BATMAX n	Restricts BATCON to run only in jobs.
:SET BATMIN n	Reserves n job slots for BATCON subjobs.
:SET CORMAX n	Simulate the monitor command SET CORMAX.
:SET CORMIN n	Simulate the monitor command SET CORMIN.
:SET DATE mmddyy	Simulate the monitor command SET DATE.
:SET DAYTIME hhmm	Simulate the monitor command SET DAYTIME.
:SET LOGMAX n	Restrict the system to run only n jobs.
:SET OPR TTY n	Simulate the monitor command SET OPR.
:SET RUN CPxn :SET RUN NO CPxn :SET RUN ONLY CPxn :SET RUN ALL	Allows the operator of a multiprocessor system to turn processors on and off line. X can be A for KA10, I for KI10, or U for either. N is the CPU number. :SET RUN adds the named processor to the system pool of running CPU's. SET RUN NO removes the named CPU from the system pool. SET RUN ONLY allows only the named CPU to run. SET RUN ALL puts all the CPU's available into the system pool. The SET RUN command requires a 5.05 or later monitor and multiprocessors.

:SET SCHEDULE n	Simulate the monitor command SET SCHEDULE.
:SET TTY arg	Simulate the SET TTY monitor command.
:SILENCE n	Ignore all output from subjob n. Reset by :REVIVE.
:SLOGIN ppn	Same as :LOGIN except suppress all the LOGIN chatter.
:STOP n	Put subjob n into monitor mode immediately by sending up to three CONTROL-C's.
:SYSTAT xx	Run SYSTAT with optional argument(s) xx over the first free subjob.
:TLOG filename.ext	Create a log file with the specified name. If the file's device is a directory one, check if the file exists already; if so, notify the operator and ask whether it should be superseded. If the answer is negative, the file will be appended to the previous file. The default filename and extension are OPSER.LOG.
:TSILENCE n	Ignore all output from subjob n (same as :SILENCE command) but place entries into the log file.
:TTYTST	Test the terminal by typing all of the ASCII characters between octal 40 and 174, inclusive.
:UNRESTRICT dev1:, dev2:; . . . ,devn:	Returns to the free user resource pool a device that had been restricted. Complement of RESTRICT. (The colon after the device name is optional.)
:WHAT n,m,p	List the status of the specified subjobs on the console. The status includes a SYSTAT including the time, the time of the last input and last output, a linear listing of the JOBSTS bits, and the time of the next times auto file.

2.1 Special Syntax

ALL may be used as the subjob specification in any command string where a subjob specification is needed; all active subjobs are implied as objects of the command.

If BATCON is running under OPSER, it should be assigned the mnemonic B. If one wishes to send text to BATCON subjob 2, he can type B2-<line> to OPSER and OPSER will send the entire line to BATCON (which will then send the part of the line after the dash to its own subjob 2). The operator can also suffix the B with any length of ALL. Refer to Chapter 4 for a further description of sending text to subjobs.

2.2 Defaults

If a subjob specification is needed but one is not present, the last subjob referenced is presumed. The use of ALL does not alter this last reference.

In the TLOG file specification, the default string is

DSK:OPSER.LOG

Absence of any field results in substitution of the default for that item. In particular, if a null extension is desired, a period, then a project-programmer specification or <CR> must be typed.

In the AUTO file specification, the default string is

DSK:OPSER.ATO

The same default rules apply to this file specification as apply to the TLOG file specification.

If an input line does not start with a dash or colon, and the first non-alphanumeric character is not a dash, that entire line is sent to the last subjob referenced.

2.3 Special Entries

The REENTER command acts exactly as a START or RUN command except that the subjob activity situation is untouched, and the low segment is not zeroed. This means the AUTO mode, if in progress, is ended, and the LOG file, if opened, is closed.

If OPSER is started at its starting Address plus one (CCL start), it looks for an auto file. The name of the auto file depends on the terminal on which OPSER is started. If it is started on the local operator's console it looks for "SYS:OPR.ATO". If OPSER is started on a remote operator's console, it looks for "SYS:OPRn.ATO" (where n is the number of the remote station). If OPSER is not started on an operator's console, it looks for "SYS:TTYn.ATO" (where n is the number of the terminal to be used to run OPSER).

3.0 SUBJOB SPECIFICATION

A subjob can be specified in any one of four ways. It can be left out entirely, in which case the last subjob referenced is presumed. One can use ALL, in which case all active subjobs are implied. One can type a decimal number from zero to the limit OPSER was generated for, or a mnemonic can be assigned to the subjob by the :DEFINE command.

4.0 SUBJOB COMMUNICATION

4.1 Input to a Subjob

The operator can send text to a subjob by typing the subjob specification, delimited by a dash, followed by the line of text. All text following the dash up to and including the break character is sent unmodified to the subjob. For example,

```
3-R MACRO<CR>
```

would result in R MACRO<CR> being sent to subjob 3. In some cases it is desirable to send many lines at once to a subjob. One should type a double dash after the subjob specification, then a delimiter, the lines, and the delimiter again. For example,

```
X--"R PIP<CR>
DSK:/X/B←DTA1:*.*<CR>
↑A DIRECT<CR>
"<CR>
```

would send all of the text between the quotes to subjob X.

```
3- foo
```

would send foo to subjob 3.

4.2 Output From a Subjob

Any output from a subjob is headed by the time of day, the subjob's name and a carriage return. Then the output is typed on the terminal.

5.0 CORE LAYOUT

With normal assembly options, OPSER is assembled in two segments; a 2K high segment, and a 1K low segment data base. All buffers except those of the AUTO device are preallocated; their sizes are subject to further assembly switches. The buffers for the AUTO device are expanded dynamically, pushing OPSER's low segment over the 1K mark. The low segment is purely block storage assignments, so no low file is written out on SAVE.

6.0 CODING CONVENTIONS

6.1 Register Assignments

OPSER's accumulators are assigned by the following set of symbols:

```
F=0             ; c (lh) = program bits
                ; c (rh) = subjob usage bits
T1=1            ; general scratch
T2=2
T3=3
T4=4
T5=5
LASU=10         ; last subjob used
MJOB=12         ; monitor job number
CMD=13          ; word input register
SJB=14          ; PTY subjob (channel) ref
PT1=15          ; byte pointer
DATA=16         ; ASCII I/O register
P=17            ; pushdown pointer
```

Registers T1 through T5 are for general scratch, to be used for LOOKUP, ENTER or other UUO's. The only time they should be preserved is in the I/O subroutines, since these routines are called so often and from so many contexts. Register PT1 must also be saved in I/O routines. During I/O error recovery routines, the ASCII byte register, DATA, must also be pushed.

6.2 Assembly Switches

MOSTBF symbolizes the maximum number of times a PTY's output buffers are transmitted before control is returned to the operator. This is to prevent loud-mouthed subjobs from taking a stranglehold on OPSER's attention.

PDLSIZ specifies how large the pushdown list should be.

SNOOZT specifies how long OPSER should sleep when there's nothing to do, or when it's in some kind of wait.

CHANCE sets the number of times OPSER will sleep waiting for a previously active subjob to resume output.

HGHPTY sets the upper subjob limit, i.e., OPSER will be generated for HGHPTY+1 subjobs, numbered from zero to HGHPTY.

FTAUTO, if non-zero, implies the assembly of the AUTO feature.

FFTLOG, if non-zero, implies the assembly of the log file feature.

DSKSIZ specifies the size of a standard disk buffer.

TTYSIZ sets the standard size of a teletype buffer.

LOGNB sets the number of pre-allocated buffers for the log file.

decsystem10

PIP

Peripheral Interchange Program

Order No. DEC-10-UPIPA-A-D

digital equipment corporation · maynard. massachusetts

1st Edition, October 1967
2nd Edition (Rev), May 1968
3rd Edition (Rev), November 1968
4th Edition (Rev), November 1969
5th Edition (Rev), June 1970
6th Edition (Rev), March 1972
7th Edition, February 1975

The HOW TO OBTAIN SOFTWARE INFORMATION page, located at the back of this document, explains the various services available to DIGITAL software users.

The postage prepaid READER'S COMMENT form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

PREFACE

The functions provided the user by the DECsystem-10 Peripheral Interchange Program (PIP) and their use are described in this manual.

NOTE

Monitor commands are available which perform the common PIP functions of copying, renaming, protecting and deleting files.

It was assumed in the preparation of this manual that the reader is familiar with or has access to the DECsystem-10 Monitor Calls manual and the DECsystem-10 Operating System Commands manual. These manuals as well as the PIP manual are available in the DECsystem-10 Software Notebooks.

CONTENTS

CONTENTS (Cont)

CONTENTS (Cont)

CHAPTER 1

INTRODUCTION

PIP (Peripheral Interchange Program) transfers files between standard I/O devices and can be used to perform simple editing and magnetic tape control operations during those transfer operations.

To call PIP into core from monitor level, the user types the command

```
.R PIP <CR>
```

When PIP is loaded and ready for input, it prints the character * at the terminal. The user may then enter the command string needed to perform the desired operations followed by a carriage return. On completion of the operation or operations requested in a command string, PIP again prints the character * to indicate that it is ready for the next command string input. To exit from PIP, the user types a Control C (↑C) command.

1.1 CONTROLLING PIP INDIRECTLY

PIP is normally controlled by commands entered via the terminal keyboard. PIP, however, is also capable of reading commands from a prepared file and executing these commands as if they had just been entered via the input console. PIP command files which are to be processed indirectly are identified by the addition of the symbol @ after the identifying file specification (see paragraph 2.1.1 for a description of file specifications). For example, the file specification FOO.CCL@ identifies the file FOO.CCL as an indirect command file. Any filename extension may be used in specifying an indirect command file. If none is given, however, the default extension .CCL is assumed.

An indirect PIP command file consists of one or more PIP commands structured as described in Section 2.

Once PIP is in core, the user passes control of PIP to an indirect command file by entering the file's filename. For example, the input command sequence

```
.R PIP <CR>
*FOO.CCL@ <CR>
```

loads PIP and initiates the execution of the indirect PIP command file FOO.CCL.

1.2 WRITING CONVENTIONS

The following symbols and abbreviations are used throughout this manual:

Symbol or Abbreviation	Meaning
dev:	Any logical or physical device name. The colon must be included when the device name is used as part of a PIP command.
file.ext	Any filename and filename extension.
[directory]	Identifies the directory of a specific file storage area within the system; it may also specify the location of a file within the identified storage area. (See paragraph 2.4 for a detailed description of [directory].) When the input terminal used is either a Model 33 or 35 Teletype unit, the right and left brackets are input in the following manner: To Obtain a: — Type: a) left bracket [— SHIFT K b) right bracket] — SHIFT M
↑ch	A control character (↑ represents the CTRL key; ch represents a character). A control character is obtained by pressing the CTRL key while typing the selected character key. A frequently used control character, for example, is ↑C, which requests that control be returned to the monitor.
=	An equals character, used in the PIP command to separate the destination and source sections.

NOTE

PIP will also accept the back arrow (SHIFT-O) entry. A SHIFT-O entry is echoed on the terminal printer as the symbol ←.

*	PIP's response to a command string to indicate that it is ready for the next input string.
'	The Monitor's response to a command string to indicate that it is ready for the next command.

<CR>	This symbol represents a carriage return, linefeed operation. It is initiated by the entry of a RETURN keyboard input. A RETURN input is normally used to terminate each PIP input command.
n	A number, either octal or decimal.
↑	This up-arrow symbol indicates the use of a CTRL key entry (see ↑ch above). Up-arrows are also used to enclose identifiers which may be assigned to DECtapes using the facilities provided by PIP (see 3.2.1.2).

CHAPTER 2

PIP COMMAND STRING AND ITS BASIC ELEMENTS

PIP command strings may be of any length; both upper and lower case characters may be used. PIP commands are normally terminated and the request operation initiated by a RETURN keyboard entry (i.e., <CR>). However, an ALT MODE (also known as ESC), line feed, vertical TAB or form feed keyboard entry can also be used as a command terminator.

2.1 COMMAND FORMAT

All PIP commands which involve the interchange (transfer) of data must have the following format:

DESTINATION=SOURCE <Terminator>

where:

1. The DESTINATION portion of a PIP command describes the device and file(s) which are to receive the transferred data. This portion of a command consists of one file specification.

2. The equals sign is a required delimiter in all PIP commands to separate the DESTINATION and SOURCE portions of the command.

3. The SOURCE side of the command describes the device from which the transferred data is to be taken. This portion of a command may contain one or more file specifications.

4. A Terminator is required to end each PIP command. A RETURN entry (symbolized as <CR>) is normally used. However, any other paper-motion command may be used as a terminator.

PIP commands which do not require the transfer of information may be written using the form

DESTINATION=<Terminator>

The equals delimiter and a terminator are still required in commands formatted in this manner despite the fact that only the destination portion of the command is used.

2.1.1 File Specification

A file specification contains all of the information needed to identify a file involved in a PIP function. It may consist of:

1. a device name;
2. a filename;
3. a directory identifier;
4. a protection code which is to be assigned to either a specified file, a User File Directory (UFD), or a Subfile Directory (SFD);
5. and an identifier to be assigned to the tape mounted on a specific DECtape unit.

The format of a PIP command containing all possible items of a file specification is:

dev:name.ext[directory]<nnn>↑ident↑=dev:name.ext[directory] <CR>

where:

1. DEV is either a physical device name (e.g., DSK, DTA1, etc.) or a logical device name (refer to paragraph 2.2).
2. NAME is a 1 to 6 character alphanumeric identification which is either to be assigned to a new file (NAME is on the destination side of the command) or which identifies an existing file (NAME is on the source side of the command). (Refer to paragraph 2.3 for a description of filenames.)
3. EXT is a 1 to 3 character alphanumeric extension assigned to the name of a file either by the user or by the system. (Refer to paragraph 2.3 for a description of filename extensions.)
4. [DIRECTORY] is the identifier of a specific directory (i.e., UFD or MFD) within the system. This identifier may consist of a project, programmer number pair and Sub File Directory (SFD) names. (See paragraph 2.4 for details.)
5. <nnn> is a 3-digit protection code which is to be assigned to either one or more destination files or to a specified User File Directory.[1] (See paragraph 2.5 for a description of protection codes.)
6. ↑IDENT↑ is a 1 to 6 character name which is to be given to the contents of a DECtape reel mounted on a specified DECtape unit. (See paragraph 3.2.1.2 for details.)

---

[1] A User File Directory (UFD) is provided by the system for each user permitted access to it. A user's UFD is identified by his project, programmer number; it contains the names of all files belonging to the user together with pointers to the actual location of each file.

The manner in which each of the possible elements of a file specification may be used in either the destination or source portions of a PIP command is described in the following table:

ELEMENT	DESTINATION	SOURCES
dev:	Name of device onto which the specified file is to be written.	Name of device on which the specified file resides.
name	Name to be assigned to the copied file.	Name of the file to be copied.
.ext	User-specified filename extension.	Current filename extension.
[directory]	Identification of the disk storage area which is to receive the file to be transferred.	Identification of the disk storage area which contains the file to be copied.
	NOTE The [directory] identifier must include a full directory path specification whenever sub-file directories are involved. For example, [proj,prog,SFDA, . ,SFDn]. (See paragraph 2.4 for more details.)	
<nnn>	Protection code to be assigned to either a copied file or a specified UFD.	NOT PERMITTED IN SOURCE PORTION OF PIP COMMANDS.
↑ident↑	Name to be assigned to the tape mounted on a specified DECtape unit.	NOT PERMITTED IN SOURCE PORTION OF PIP COMMANDS.

File specifications may be delimited by:

1. an equals character (=) if the specification is on the destination side of the command string (e.g., dev:name.ext=. . .<CR>).

NOTE

PIP will accept a back-arrow entry (←) in place of the equals character (=).

2. a comma (,) if the specification is on the source side of the command string and is one of a series of file specifications. For example

   dev=dev1:name.ext,dev2:name.ext,name.ext, . . .name.ext<CR>

3. a RETURN <CR> entry if it is the last item on the source side of a command. For example

   dev=dev1:name.ext,dev2:name.ext, . . .devn:name.ext<CR>

2.1.2 Command String Delimiters

The delimiters which may be used to separate the elements of a PIP command string are described in the following table.

PIP Command String Delimiters

DELIMITER	USE AND DESCRIPTION
:	The colon delimiter follows and identifies a device name. For example, the device DTA1 is specified as DTA1: in PIP commands.
[]	Square brackets are used to enclose the user DIRECTORY numbers and SFD names (if SFDs are used). For example, [40,633] or [40,633,SFD1, SFD2, . . .SFDn] represent the manner in which DIRECTORY numbers can be written.
<>	Angle brackets must be used to enclose a protection code (e.g., <057>) which is to be assigned to either a file or a user file directory (UFD) or subfile directory (SFD).
,	Commas are used to separate user project and programmer numbers, SFD names, and file specification groups. For example dev:[40,633]=dev:name.ext,name.ext<CR>
↑↑	A name to be assigned as an identifier to a DECtape is enclosed within a set of up-arrows (e.g., .↑MACFLS↑).
.	A period delimiter must be the first character of a filename extension. The form of an extension is .ext.
#	A number symbol is used as a flag to indicate the presence of an octal constant in a filename or a filename extension.

PIP Command String Delimiters (Cont.)

DELIMITER	USE AND DESCRIPTION
!	An exclamation symbol may be used to delimit a file specification. When used, the ! symbol causes control to be returned to the monitor from PIP and the specified file (or program) to be loaded and run. This function is provided as a user convenience to eliminate the need for several control entries.
=	The equals character must be used to separate the destination and source portions of a PIP command.
( )	Parentheses are used to enclose magnetic tape options, PIP control switches, and one or more PIP function switches. The form of a command employing parentheses to enclose a series of switches is: dev:name.ext(/sw(1)/sw(2), ./sw(n)=. . . <CR>
/	The slash is used to indicate and to separate switch designations.
@	When there are many program names and switches, the at sign is used to put them into a file so that the user does not have to type them for each compilation.

2.2 DEVICE NAMES

Both physical and logical device names may be used in PIP commands. The user must remember that a logical name takes precedence over a physical name which is the same as the logical name.

2.2.1 Physical Device Names

Each standard DECsystem-10 peripheral device is assigned a specific device name consisting of a 3-character generic name plus a unit number (0 to 777) or:

1. 3 characters,
2. 3 characters and a station number,
3. an abbreviated disk name, or
4. the name or a disk file structure.

A list of the generic physical device names is given below:

Peripheral Devices

DEVICE	GENERIC PHYSICAL DEVICE NAME
Card Punch	CDP
Card Reader	CDR
Console TTY	CTY
DECtape	DTX
Disk	DSK
Packs	DPX
Fixed-Head	FHX
Display	DIS
Line Printer	LPT
Magnetic Tape	MTX
Operator Terminal	OPR
Paper-tape Punch	PTP
Paper-tape Reader	PTR
Plotter	PLT
Pseudo-TTY	PTY
System Library	SYS
Terminal	TTY
Pseudo-device TMPCOR	TMP

2.2.2 Logical Device Names

A logical device name is a user-assigned designation which is employed in the preparation of a program in place of a specific physical device name. The use of logical device names permits the programmer to write programs which do not specify one particular device but may use, at run time, any available device which can perform the required function.

A logical device name may consist of from one to six alphanumeric characters of the user's choice.

2.3 FILENAMES

Filenames are file identifiers assigned either by the system (for system programs) or by the user. A filename may consist of a name field and an extension field but only a name field is required. Whenever both fields are used in a filename, it has the form name.ext. A period delimiter is required as the first character of the extension. Filename fields are defined as:

1. Name Field. Names of files may consist of from one to six alphanumeric characters or octal constants; in user-assigned names the characters may be arbitrarily selected by the user. Names generated by the user must be unique at least within the file structure and directory number in which the file is located.

2. Extension Field. Filename extensions may consist of up to three alphanumeric characters. Extensions are normally used to specify the type of data contained by the file identified by the filename field. Filename extensions which are recognized by the system, and the type of data each specifies are given in Appendix A. In filenames, a user may specify a standard extension (one recognized by the system), one which he has devised, or none at all.

   PIP utilizes the filename extension given in a file specification to determine whether the file is to be transferred in a binary or ASCII mode. If it is at all possible, PIP will transfer files in a binary mode since it is faster.

   In dealing with filename extensions, PIP performs a specific series of tests in order to determine the mode which should be used during a requested transfer operation. The following mode determination tests are performed in succession until PIP obtains a firm indication as to the type of mode required:

   a. PIP tests for the presence of a data mode switch (see paragraph 3.4). If no switch is found, PIP goes to the next test.

   b. PIP tests for the presence of a known (standard) filename extension which specifies a binary mode of transfer (see Appendix A). If no binary extensions are found, PIP goes to the next test.

   c. PIP tests both the input and the output devices specified to determine if they are both capable of handling binary data. If either or both of the devices cannot handle binary, the transfer is made in the ASCII mode. If both devices can handle binary data, PIP goes to the next test.

   d. PIP tests for the presence of an X option switch (/X) in the command string; if it is found, the transfer is made in the binary mode. If an X option is not found, PIP goes to the next test.

   e. PIP tests for the presence of commas (non-delimiters) in the command string; if commas are found an ASCII mode is indicated. If no commas are found, the transfer is made in the binary mode.

2.3.1 Naming Files with Octal Constants

Octal constants may be used as either a part of or all of a filename. In either of the foregoing cases, each octal constant which appears in a filename must be preceded by the symbol #, and each group is delimited by a non-octal digit or a character. For example, the filenames:

1. #124ABC.ext (constants are used as part of a filename)
2. #12AB#34.ext (constants are intermixed with other characters)
3. #124670672346.#123737 (constants from the whole filename)

are all acceptable to PIP.

The symbol # is not regarded by PIP as part of the filename but is used only to indicate an octal constant.

The number of octal digits used in a filename or an extension should be even since two octal digits may be stored in a SIXBIT character. If an odd number of octal digits is given, PIP will add an extra 0 to the filename or extension. For example, the constant #123 would be expanded to #1230 by PIP.

Names comprised of octal constants are left-justified by PIP. The following is an example of the use of octal filenames:

```
DTA01:#124670.BIN=DSK:#100000.BIN<CR>
```

2.3.2 Wildcard Characters

The two symbols * and ? may be used in PIP to represent, respectively, complete fields and single characters. These symbols are referred to as wildcard characters; their use is described in the following paragraphs.

2.3.2.1 The Asterisk Symbol – The asterisk symbol * may be used to replace a filename or extension:

1. name field (e.g., *.ext),
2. extension field (e.g., name.*),
3. both filename fields (e.g., *.*).

For example, the filename FILEA.MAC, which specifies the MACRO source language file named FILEA, may be altered by the use of the asterisk in the following manner:

1. *.MAC specifies all files with the extension .MAC.
2. FILEA.* specifies all files with the name FILEA, and,
3. *.* specifies all files.

2.3.2.2 **The Question Mark Symbol** – The character ? may be used to indicate a wild character in file names and extensions. A ? in a file name allows any character (or none, if ? is the last character in a field) to match or be inserted for the ? when the file name is used. This masking capability enables the user to specify, with one command, groups of files whose filenames have common characters, identically positioned. For example, assume that the device DTA1 contains the files TEST1.BIN, TEST2.BIN, TEST3.BIN, and TEST4.BIN; the user can specify all of these files with one file specification:

```
DTA1:TEST?.BIN
```

2.3.2.3 **Combining * and ? Wildcard Symbols** – The symbols * and ? can be combined in filenames to specify groups of files which have common characters in either or both of their names or extensions. For example, the file specification

```
ABC???.*
```

specifies all files having the character group ABC as the first three characters of their filenames. Again, the file specification

```
*.??A
```

specifies all files having an extension whose third character is A.

In combining the * and ? symbols, the user should remember:

1. for filenames, * is equivalent to ??????, and
2. for extensions, * is equivalent to ???.

For example, the filenames *.* and ??????.??? are equivalent.

2.4 DIRECTORY IDENTIFIER

The [directory] identifier is used in PIP commands to identify a specific:

1. User File Directory (UFD),
2. Sub File Directory (SFD), or
3. a specific UFD-SFD directory path.

The item identified by a given [directory] identifier can be a directory or an item located within a directory which belongs to either the current user or, when the protection code scheme permits, to another user. (See paragraph 2.5 for a description of protection codes.)

A [directory] identifier can consist of a project,programmer number pair (abbreviated as proj,prog) and the names of SFDs. The most expanded form of the [directory] identifer is:

[proj,prog,SFD1,SFD2, . . .SFDn]

As shown, a [directory] identifier is always enclosed within square brackets and its elements are delimited by commas.

2.4.1 UFD-Only Identifiers

Each UFD is identified in the system by the project,programmer number pair assigned to the user for whom the UFD was created. A [directory] identifier for a UFD has the form

[proj,prog]

UFD [directory] identifiers may be written without either one or both of the project,programmer numbers. In such cases, PIP assumes either a previously specified default number or the number assigned to the current user. For example, assume that the current user is logged in under the number pair [57,124] and that no default identifier has been specified. The current user can use [directory] identifiers having any of the following formats:

	The Format:	Which is Interpreted by PIP as:
1.	[,]	[57,124]
2.	[57,]	[57,124]
3.	[,124]	[57,124]

2.4.2 SFD (Full Directory Path) Identifiers

A Sub File Directory (SFD) is identified by its user-assigned name plus the project,programmer number pair which identifies the UFD in which it is located. A [directory] identifier for an SFD then has the form

[proj,prog,SFDname]

Whenever an SFD is located in a UFD which has a multi-level directory arrangement, the UFD containing the desired SFD must be included in the [directory] identifier for the desired SFD. A [directory] identifier for an SFD in a multi-directory level UFD has the form

[proj,prog,SFD1,SFD2, . . .SFDn]

and is referred to as a full directory path identifier. For example, assuming that the current UFD is identified by the proj,prog number pair 57,124 and has the following directory organization:

Level 1		UFD	
Level 2		SFDA	
Level 3	SFD1		SFDB
Level 4	SFD2		SFDC

the [directory] identifier for SFD2 is written as

[57,124,SFDA,SFD1,SFD2]

The proj,prog number pairs in full directory path identifiers may be written using the format variations described in paragraph 2.4.2. However, when no proj,prog numbers are specified by the user, two commas must be used in the identifier in the following manner

[, ,SFD1, . . .SFDn]

The first comma represents the delimiter between the proj,prog numbers; the second represents the delimiter between the last number (prog) and the first SFD name.

2.4.3 Specifying Default and Current [Directory] Identifiers of Source Files

The position in which a [directory] identifier is given in a PIP command determines if it is viewed as a default identifier for all subsequent file specifications given in that command or is the current identifier for an individual file specification.

If a [directory] identifier is given before one or more file specifications of a command, it is regarded as the DEFAULT identifier for those specifications. For example, in a command segment having the form:

[directory A] File Specification 1,File Specification 2

the identifier [directory A] is the default for both File Specifications 1 and 2.

If a [directory] identifier is given after the filename within a File Specification, it is viewed as the current identifier for that file specification and will override any given default [directory]. The form of a file specification with the current identifier specified is:

dev:filename.ext[directory]

Both default and current [directory] identifiers can be specified in the same PIP command. For example, the PIP command source segment:

=dev:[directoryA]filename.ext,dev:filename.ext[directory B] <CR>

is valid. In the foregoing example, the identifier [directory A] is the default identifier for the first file specification, and will act as the default identifier for the second file specification. If [directory B] is not given. When [directory B] is given, it overrides the default identifier and is accepted as the identifier for the second file specification.

2.5 FILE ACCESS PROTECTION CODES

Three-digit (octal) protection codes which specify the degree of access that each of three possible types of users may gain to a file can be specified in the destination side of a PIP command string. File access protection codes are written within angle brackets and must contain three digit positions (e.g., <nnn>). Each digit within a protection code specifies the type of access a specific type of user may have to the file or files involved. Considering the protection code <n1n2n3> the digits give the file access code for the following types of users:

1. n1 = File OWNER,
2. n2 = project MEMBER, and
3. n3 = OTHER system users.

The user types are defined as follows:

1. FILE OWNERS. Users who are logged in under either:
   a. the same programmer number as that of the UFD which contains the file; or,
   b. the same project and programmer number as associated with the UFD which contains the file.

   The decision as to which of the above items defines an OWNER is made at Monitor Generation time.
2. PROJECT MEMBER. Users who are logged in under the same project number as that which identifies the UFD containing the file.
3. OTHER USERS. Any user of the system whose project and programmer number do not match those of the UFD containing the file in question.

File access protection codes are placed in PIP commands after the destination filename of the file involved. For example, the command:

DPA3:FILEA.BIN<nnn>=DSK:SOURCE.BIN<CR>

copies the contents of file SOURCE.BIN onto disk pack DPA3 under the name FILEA.BIN with an assigned file protection code of nnn.

2.5.1 Digit Numeric Protection Code Values

Each of the digits in a 3-digit file protection code may be assigned an encoded numeric value ranging from 0 to 7. The meaning of each octal value is:

CODE VALUE	PERMITTED OPERATIONS
7	No access privileges. File may be looked up if the UFD permits.
6	Execute only.
5	Read, execute.
4	Append, read, execute.
3	Update, append, read, execute.
2	Write, update, append, read, execute.
1	Rename, write, update, append, read, execute.
0	Change protection, rename, write, update, append, read, execute.

Files are afforded the greatest protection by the code value 7; the least protection by 0. It is always possible for the owner of a file to change the access protection associated with that file even if the owner-protection field is not set to 0; thus, the value 0 and 1 are equivalent for the owner. Files with their owner-protection field set to 1 are preserved (i.e., saved by .KJOB/K).

It is recommended that important files such as source files be assigned an owner-protection code of 2. This level of protection will prevent the files from being accidentally deleted while permitting them to be edited.

2.6 UFD AND SFD PROTECTION CODES

When a user directory (UFD or SFD) is created, it is assigned a 3-digit octal access protection code by either the owner of the file or, by default, the system. The 3-digit code specifies the type of access permitted to the directory by each of the three possible classes of users (i.e., OWNER, MEMBER, or OTHER). (See paragraph 2.5 for a description of user classes.)

Once assigned, a directory access protection code may be changed by the owner and, if the protection code permits (i.e., CREATES allowed), by users other than the owner. (See the description of the PIP rename option given in paragraph 3.5.3.1 for the procedure required to change directory protection codes.)

The access protection code assigned each user class may range from 0 through 7; the following table lists the codes and the operations which each permits.

CODE	PERMITTED OPERATIONS (S)
0	Access not permitted.
1	The directory may be read as a file.
2	CREATEs are permitted.
3	The directory may be read as a file and CREATEs are permitted.
4	LOOKUPs are permitted.
5	The directory may be read as a file and LOOKUPs are permitted.
6	CREATEs and LOOKUPs are both permitted.
7	The directory may be read as a file and both CREATEs and LOOK-UPs are permitted.

CHAPTER 3

STANDARD PIP SWITCHES

PIP provides the user with a group of optional functions which can be executed during the performance of the primary PIP transfer function.

Each optional function is assigned an identifier which, when added as a "switch" to a PIP command, initiates the execution of the identified function.

For the purposes of this manual, the PIP optional functions are divided into standard and special groups. The standard group of options described in this section consist of switches which:

1. determine which files are transferred;
2. edit all the data contained by each source file;
3. define the mode of transfer;
4. manipulate the directory of a directory-type device.

All optional functions which deal with non-directory devices and which perform functions other than those listed above are considered special and are described in Section 4.

3.1 ADDING SWITCHES TO PIP COMMANDS

All switches in PIP commands must be preceded by a slash (i.e., /sw) or enclosed within parentheses (i.e., (sw)); for example, the optional function identified by the letter W is added to a PIP command:

```
*DTA1:DESTFL.BIN/W=DSK:FILEA.BIN,FILEB.BIN<CR>
```

When more than one switch is to be added to a command, they may be listed either separated by slashes (e.g., /B/X....) or enclosed in parentheses (e.g., (BX)).

3.2 BASIC TRANSFER FUNCTION

The basic function performed by PIP is the interchange (i.e., read/write transfer) of files or data blocks between devices. There are two types of transfer operations:

1. An optional X-switch transfer in which the source files or blocks are transferred as separate files to the destination device.

2. A non-X type in which all files or blocks transferred from the source device are combined (i.e., concatenated) into a single file on the destination device.

3.2.1 X-Switch Copy Files Without Combining

The use of the X-switch enables the user to move (copy) a group of source files onto the destination device as individual files without changing their creation dates, creation times, filenames and filename extensions. The following are examples of how the X-switch is used in PIP:

1. To transfer all the user's disk files to a DECtape, type:

   DTA1:/X=DSK:*.*<CR>

   Assuming that there are three files on the user's disk area named FILEA, FILEB, FILEC.REL, these files will be transferred to DTA1 and can be referenced on DTA1 by those names.

   One significant difference between the disk and all other devices is file protection. If the disk is the source device, PIP will by-pass those protected files to which the current user is not permitted access. A suitable message is then issued by PIP if the rest of the command string is successfully executed. Similar processing is described later for the L, Z and D switches. If none of these switches is given, a requested DSK file which is protected will cause termination of the request.

2. To transfer all the files from card reader to disk, type:

   DSK:/X=CDR:*<CR>

   When files are transferred from the card reader with the * command, the input files must either be wholly ASCII or wholly binary.

3. To transfer two specific files from user [11,7]'s disk area to a DECtape, type:

   DTA2:/X=DSK:[11,7]FILEA.REL,FILEA.MAC<CR>

4. To copy files from a paper tape onto a directory-type device, the user may employ either:

   a. A copy command in which the number of files to be read is specified by adding a series of commas to the command after the source device name (i.e., PTR:,,,,,,). The number of commas required is always one less than the total number of files to be transferred. For example, the command:

      DSK:/X=PTR:,,,,<CR>

specifies that five (5) files are to be copied from paper tape and written, individually, into the current user's disk area.

b. A copy command in which the file on a paper tape is to be copied onto a specific device. For example, the command

DSK:/X=PTR:<CR>

specifies that the file on the paper tape in the PTR is to be copied into the current user's disk area. Whenever a command of this type is used, the last file on the paper tape must be followed by two consecutive end-of-file codes.

NOTE

In both the foregoing examples, PIP will generate any needed destination filenames. This function is described in paragraph 3.2.1.1.

Whenever the X-switch is used and is not combined with an editing option, PIP transfers any file involved as it appeared on the source device. X-switch operations are copy operations and are referred to as such.

3.2.1.1 **Non-Directory to Directory Copy Operation** – In copying files from a non-directory device onto a directory-type device, PIP must perform special operations in naming the destination files. For example, a special case of source and destination filenames arises in the command:

DTA2:FNME.EXT/X=MTA0:*<CR>

Here, every file is to be copied from a non-directory device (MTA0) to a directory device (DTA2) without combining files (/X). Only one destination filename is given (i.e., FNME.EXT) but the source device (MTA0) may contain more than one file. If more than one file is transferred, it is necessary for PIP to generate a unique filename for each copied file. PIP generates filenames by developing a 6-character name field in which the first three characters are either:

1. the first three characters of a given destination filename, or

2. the characters "XXX" if no destination filename is given in the command.

The second portion of the PIP-generated name field consists of the decimal numbers 001 through 999 which are added, in sequence, to each filename developed during the /X copy operation.

For filename extensions, PIP uses either the extension of a given destination filename or a null field if no filename is given in the command.

For example, assuming that three files are present on MTA0, the command:

DTA2:FNME.EXT/X=MTA0:*<CR>

transfers the files to DTA2 and establishes the following names in the DECtape directory for the files copied:

1. FNM001.EXT,
2. FNM002.EXT,
3. FNM003.EXT.

If, in the above example, the command given did not include a destination filename (i.e., DTA2:/X=MTA0:*<CR>) the copied files would have been named:

1. XXX001
2. XXX002
3. XXX003

The use of the 3-digit decimal number for the last three characters of the filename name gives the user 999 possible input files from non-directory devices. If PIP finds more than 999 files on the source device, it will terminate the transfer operation after the 999th file is copied and will issue the error message

?TERMINATE/X,MAX OF 999 FILES PROCESSED.

Any error message referring to individual files named by PIP (either input or output) will use the generated filename.

3.2.1.2 Assigning Names to DECtape Tapes – A tape mounted on a specified DECtape unit can be assigned a label during copy operations. A labels are from 1 to 6 character names (any SIXBIT character – except ↑ – within the code range 40-137 can be used) which are added to the DECtape's directory (128th word). DECtape identifiers can be read by the PIP, FILEX and DIRECT programs; the monitor does not read identifiers. A DECtape identifier is assigned by adding the selected name to a PIP command when the DECtape to be named is mounted on the specified destination device.

The format required for a DECtape identifier is

↑name↑

A DECtape identifier is inserted into a PIP command following the given destination device name:

dev:↑name↑=source file specification(s)

For example, the command

```
*DTA3:↑MYFILE↑/X=DTA1:*.*
```

specifies that the DECtape on device DTA3 be given the identifier "MYFILE" and receive copies of all the files contained by the tape on device DTA1.

3.2.2 (DX) Copy All But Specified Files

When the (DX) is added to a PIP command it causes all the files to be copied from the source device to the destination device except those files which are named in the command string. If the source device is DSK, a maximum of 10 source-file specifications is allowed. Only directory-type devices are allowed as source devices; no check is made on the existence of the files which are not to be copied. Only one source device is permitted; for example, the command

```
DTA1:(ZDX)=DSK:*.LST,*.SAV,CREF.CRF<CR>
```

zeroes out the directory of DTA1 and transfers to DTA1, from the disk, all files except CREF.CRF and all files with an extension of .LST or .SAV.

3.2.3 Transfer Without X-Switch (Combine Files)

When the X-switch is not included in a PIP command all files or blocks transferred from the source device are combined into a single file on the destination device. For example:

1. To combine three paper tape files into one, type

   ```
   PTP:=PTR:,,<CR>
   ```

2. To combine two files on DECtape into one on another DECtape, type

   ```
   DTA3:FILCOM=DTA2:FILA,FILB<CR>
   ```

3. To combine files from two DECtapes into one on the user's disk area, type

   ```
   DSK:DSKFIL=DTA2:ONE,DTA4:TWO,MAC<CR>
   ```

4. To combine all the files on MTA0 into one file on the user's disk area, type

   ```
   DSK:TAPE.MAC=MTA0:*<CR>
   ```

   (This assumes that MTA0 is positioned at the Load Point.)

3.2.4 U-Switch, Copy DECtape Blocks 0, 1 and 2

The U-switch is used during DECtape-to-DECtape copy operation to specify that blocks 0, 1 and 2 of the source tape are to be copied onto the destination tape.

This switch is commonly used to transfer DTBOOT from one tape to another. For example, the command:

```
DTA1:/U=DTA5:<CR>
```

transfers blocks 0 through 2 of DTA5 to DTA1.

3.3.1 A-Switch, Integral Output Lines (Line Blocking)

The use of the A-switch (/A) in a PIP command specifies that each output buffer is to contain an integral number of lines: no lines are to be split between physical output buffers. Line blocking is required for FORTRAN ASCII Input. Each line starts with a new word.

3.3.2 C-Switch, Delete Trailing Spaces and Convert Multiple Spaces to Tabs

The addition of a C-switch (/C) to a PIP command causes groups of multiple spaces in the material being copied to be replaced by one or more TAB codes; trailing spaces are deleted.

The conversion of the spaces to TAB codes is performed in relation to the standard line TAB "stop" positions located at 8-character intervals throughout the line. Only those groups of multiple spaces which precede a TAB "stop" will produce a TAB code. For example:

1. [space][stop][text of line]--will not produce a TAB code.

2. [space][space][stop][text of line]--will produce [TAB].

3. [space][space][stop][space][space][text of line]--will produce [TAB] [space][space][text of line]

A totally blank input is replaced by one space when this switch is used. The C-switch is used to save space when storing card images in DSK file structures. The conversion of spaces to tabs must be done with care since it could alter Hollerith text.

3.3.3 E-Switch, Ignore Card Sequence Numbers

This switch, normally used when a card reader is the source device, causes characters (i.e., columns) 73 through 80 of each input line to be replaced by spaces.

3.3.4 N-Switch, Delete Sequence Number

This switch causes line sequence numbers to be deleted from any ASCII file being transferred. Line sequence numbers are recognized as any word in the file in which bit 35 is a binary 1 and follows a carriage return, vertical TAB, form feed for start-of-file identification. Nulls used to fill the last word(s) of a line are ignored. If a line sequence number is followed by a TAB, the TAB is also deleted.

3.3.5 S-Switch, Insert Sequence Numbers

This switch causes a line sequence number to be computed and inserted in the output buffer at the start of each line. Sequence numbers are indicated by a 1 in bit 35 of a word following a carriage return, a vertical TAB or start-of-file indicator. A TAB is added as the first character following the line sequence number. Any previous line sequence numbers and their TAB's are removed.

Sequence numbers assigned by PIP take the form nnnnn, starting at 00010 and ranging through 99990 in increments of 10. Approximately one-third of each output buffer is left blank to facilitate editing operations on the file (DTA only).

3.3.6 O-Switch, Insert Sequence Numbers and Increment By 1

This switch causes the same operations to be performed as those for switch S, (see 3.3.5) except that the assigned sequence numbers are incremented by 1 instead of 10.

3.3.7 P-Switch, Prepare FORTRAN Output for Line Printer Listing

This switch causes PIP to take output generated by a FORTRAN program, which was output on a device other than the line printer (LPT), for which it was intended, and performs the carriage control character interpretations needed when the data is sent to the LPT. The first character in each input line is interpreted by PIP according to the following table.

FORTRAN Carriage Control Character Interpretation

CARRIAGE CONTROL CHARACTER PRODUCED BY FORTRAN PROGRAM	ASCII CHARACTER(S) SUBSTITUTED	LINE PRINTER ACTION
space		Skips to next line (single space) with a FORM FEED after every 60 lines.
*	023	Skips to next line no FORM FEED.

FORTRAN Carriage Control Character Interpretation (Cont'd)

CARRIAGE CONTROL CHARACTER PRODUCED BY FORTRAN PROGRAM	ASCII CHARACTER(S) SUBSTITUTED	LINE PRINTER ACTION
+	015	Precede line with a carriage return only (i.e., over-print previous line).
, (comma)	021	Skips to next 1/30th of page.
=	015,012,012	Skips two lines.
.	022	Skips to next 1/20th of page.
/	024	Skips to next 1/6th of page.
0	015,012	Skips 1 line (double space).
1	014	Skips to top of next page (page eject).
2	020	Skips to next 1/2 page.
3	013	Skips to next 1/3 page (also vertical tab).

3.3.8 T-Switch, Delete Trailing Spaces

This switch causes all trailing spaces to be deleted from the file being transferred. If a transfer line consists of nothing but spaces, then a single space and a line terminator will be retained in its place in the copied file.

3.3.9 W-Switch, Convert Tabs to Spaces

The addition of a W-switch (/W) to a PIP command causes each TAB code contained by the material being copied to be converted to one or more sequential spaces.

The number of spaces produced when a TAB code is converted is determined by the position of the TAB in relation to the standard line TAB "stops." Each line has TAB stops positioned at 8-character intervals throughout the length of the line. When a TAB is converted in a /W switch operation, only enough spaces are produced to reach the next sequential line TAB stop position. For example, the series

[stop]ABCD[TAB]

is converted to

```
[stop]ABCDspspspsp[stop]
```

where:

sp = space.

The use of the W-switch causes files previously edited by a C-switch to be restored to their original form (less the deleted trailing spaces and any TAB's which were in the original file).

3.3.10 V-Switch, Match Angle Brackets

This switch is not a true edit switch, because the input file is not edited. The use of this switch generates an output file which contains the results of cumulative matching of angle brackets located in the input file. If a line in the input file contains brackets which are not needed to match earlier brackets and which match each other, no output occurs. In all other cases where brackets occur, a cumulative total and the line currently considered are printed. The symbol > scores a negative count; the symbol < scores a positive count. A typical use for this switch is to check source input to the MACRO-10 Assembler; for example, assuming that the file A contains:

```
ONE<<>
TWO<
THREE>
FOUR<>>
FIVE<>
SIX>
```

The request

```
LPT:=DTA2:A/V<CR>
```

results in the Line Printer output:

```
 1 ONE<<>
 2 TWO<
 1 THREE>
 0 FOUR<>>
-1 SIX>
```

From this general example, the most likely conclusion is that there is either a < missing or an extra > in this file. Line five (i.e., FIVE <>) was not printed because the brackets which it contained were matched.

3.3.11 Y-Switch, DECtape to Paper Tape

The Y-switch enables the user to transfer DECtape files having the filename extension .RMT, .RTB or .SAV onto SAVE-formatted RIM10 or RIM10B paper tapes. The type and contents of the paper tape produced in a Y-transfer are determined by the source file filename extension. If the extension is:

1. .RMT, – A RIM10 paper tape (with terminating transfer word) is produced;

2. .RTB, – A RIM10B paper tape (with RIM loader and terminating transfer word) is produced;

3. .SAV, – A RIM10B paper tape is produced (with neither RIM loader nor terminating transfer word).

For example, the command

PTP:/Y=DTA2:TEST1.RTB<CR>

will punch a RIM10B tape as described in item 2 of the foregoing description from DECtape file TEST1.RTB.

Switches D and X may be used in conjunction with the Y-switch.

It is assumed that .RTB, .RMT and .SAV files are all in the standard "save" file format. In particular, it is assumed that no block of an .RMT saved file overlaps a preceding one.

NOTE

Optional switch Y is obtained by setting RIMSW=1 at assembly time (see source file PIP.MAC.).

The functions performed by PIP during /Y transfers in response to each possible type of source file filename extension are as follows:

1. An .RTB file causes PIP to:

   a. Punch a RIM loader.

   b. Punch an I/O word (-n,x) at the start of each data block. The variable n is the number of data words punched in each block and has the octal value 17, or less. The variable x is the starting address-1 for loading the following data. Successive values of x are derived from the pointer words in the DECtape blocks. The first value of x is the value of the right side of the first pointer word in the DECtape file.

   c. The complete DECtape file is punched as described in Item b.

d. The final block punched is followed by a block containing a transfer word. If the right half of .JBSA contains 0 then a halt is punched. If the right half of .JBSA contains a non-zero value, a jump to that address is punched.

2. A .SAV file is treated in the same way as one having an .RTB extension except that no RIM loader and no transfer word are punched.

3. An .RMT file initiates PIP functions which are similar to those described for .RTB files but which have the following differences:

   a. Only one IOWD is produced, (-n,x) where (n-1) data words and a transfer instruction follow.

   b. The first of the (n-1) data words punched from the saved file is the first word of the logical block which contains location .JBDA (i.e., the first location after the end of the JOBDATA area).

   c. The variable x is then set to the starting address (address-1) of the first data word found. The effective program length is determined by the relationship n=(.JBFF)-x. Data is now transferred from (x+1) until (n-1) words have been punched.

   d. Zero fill is used if a pointer word in a source block indicates noncontiguous data. The transfer word, calculated as described for .RTB files, terminates the output file.

3.4 SET DATA MODE, SWITCHES B, H AND I

The addition of optional data mode switches to a PIP command specifies the mode in which the file(s) involved must be transferred.

Data modes are device dependent; complete descriptions of their use and effect on different devices are given in the DECsystem-10 Monitor Calls manual.

In both input and output devices can do binary I/O, no editing switches are in force, and no concatenation is required, then all files are transferred in binary mode (36-bit bytes). If an editing switch that requires PIP to do character processing is used, ASCII mode is used. The data mode switches are:

1. /B – Initializes the input and output devices in binary mode.

NOTES

Since PIP recognizes the following as binary extensions, /B is not required when these extensions are used in the PIP command.

Binary Extensions Recognized by PIP

.BAC	.LOW	.RTB
.BIN	.MSB	.SAV
.BUG	.OVR	.SFD
.CAL	.QUC	.SHR
.CHN	.QUE	.SVE
.DAE	.REL	.SYS
.DCR	.RIM	.UFD
.HGH	.RMT	.XPN

2. /H – Initializes the input and output devices in image binary mode.

3. /I – Initializes the input and output devices in image mode.

3.5 FILE DIRECTORY SWITCHES

Optional PIP switches whose functions affect user file directories are described in paragraphs 3.5.1 through 3.5.6.

3.5.1 L-Switch, List Source Device Directory

NOTE

The Monitor command DIRECT provides the user with more facilities for obtaining directory-type information than the PIP L-Switch option. (See the DECsystem-10 Operating System Commands Manual for details.)

This switch enables the user to obtain a listing of the source device directory. The type of output device used affects the directory as follows:

1. If the output device is TTY, the directory listing formats for directory-type devices are:

   a. For DTA source (e.g., TTY:=DTA4:/L<CR)

```
TAPE ID:
FREE:  n BLKS, m FILES
filename.ext    no. of blocks   creation date
      .
      .
      .
      .
```

b. For DSK source (e.g., TTY:=DSK:/L<CR>)

filename.ext no. of blocks <protection> creation date
.
.
.
.
Total Blks n

Asterisk or question mark wildcard symbols (see paragraph 2.3.2.2) can be used in either the specified filename or extension fields to cause only those files in the disk directory of a particular filename or extension to be listed. Thus, the command TTY:/L=DSK:*.REL<CR> causes only those files with extension .REL to be printed in the directory listing.

2. If the output is not TTY, the directory listing is printed in one of the following formats:

   a. For DTA source, format is as in paragraph 1.(a)

   b. For DSK source, format is as in paragraph 1.(b) but includes access date and mode as well as the creation time and access date. If any disk file is protected, as much information as possible is given about it.

3.5.2 F-Switch, List Limited Source Directory

This switch performs essentially the same function as the L-Switch; however, only the filenames and extensions of the files in the specified disk or DECtape directory are listed.

NOTE

The Monitor command DIRECT provides the user with more facilities for obtaining directory-type information than the PIP F-switch option. (See the DECsystem-10 Operating System Commands Manual for details.)

Only disks and DECtapes are permitted as source devices; if no source device is given, DSK or TMP is assumed.

For example, the command

TTY:/F=<CR>

lists the directory of the user's disk area as described. The /F switch may work in cases where /L will not because of file access protection.

3.5.3 R-Switch, Rename Source Files

The use of this switch causes PIP to rename the source file to the name given as the destination file name. Only one source file specification can be given. If more than one is given, the error message ?PIP COMMAND ERROR is printed and no action is taken. Wildcard characters are valid in destination file specifications. Protection codes <protection> can always be specified; in fact, the wildcard request *.* has no effect without <protection>. If no protection is specified, the current file protection is not altered.

During a rename operation on device DSK, if PIP finds that the filename to be changed exists on more than one file structure, PIP will output the following message to the user's terminal:

?AMBIGUOUS[file structure list] [filename.ext]

The following are examples of the proper use of the R-switch:

1. DSK:MONI.F4/R=MONI.MAC<CR>

   Rename the file MONI.MAC as MONI.F4.

2. DSK:MON2.*/R=MONA.*<CR>

   Rename all files of name MONA and any extension to retain the extensions but take the new name MON2.

3. DSK:*.EXT/R=*.MAC<CR>

   Rename all files of extension MAC to retain their own names but take the extension EXT.

4. DSK:*.*<077>/R=*.SAV<CR>

   Give all files of extension SAV the protection <077>.

5. DTA1:MON2/R=MONA.REL<CR>

   Rename the file MONA.REL to have the name MON2 and the null extension.

3.5.3.1 Changing Source UFD or SFD Protection Code Using the Rename (R) Function – The access protection codes assigned to UFDs or SFDs can be changed using the PIP rename switch (/R) if the privileges assigned the current user permit the operation. (For the details of user UFD and SFD access privileges, see 2.4 and 2.6; see also the DECsystem-10 Monitor Calls Manual.) The owner of a directory is always permitted the use of the PIP rename function.

The command format required to change a directory access protection code is

*dev:[directory].UFD<nnn>/R=[directory].UFD<CR>

where:

1. <nnn> represents the desired (new) protection code.

2. [directory] must be the same on both sides of the command.

3. The user indicates to PIP that the protection code of the identified directory (UFD or SFD) is to be changed by specifying the extension .UFD without a filename. Note that the same extension, .UFD, is used when changing the access protection of an SFD as well as for changing the protection of a UFD.

The following examples illustrate the use of the R-switch in changing the access protection codes of directories.

1. The command:

   DSKA:[57,123].UFD<222>/R=[57,123].UFD<CR>

   changes the access code of the UFD identified by the number pair 57,123 to 222.

2. The command:

   DSKA:<222>/R=111.SFD[57,123,AAA,BBB]<CR>

   changes the access code of the SFD named 111 to the value 222. Note that 111.SFD is a file contained in [57, 123, AAA, BBB].

3.5.3.2 Changing Directory Using R-Switch – A directory specification in a rename command is an exception to command scanning conventions. If one directory specification is given, it is used for both source and destination. To change the directory, both source and destination specifications must be given. This exception prevents undesired directory changes.

3.5.4 D-Switch, Delete Files

This switch causes PIP to delete one or more specified files from the device given in the destination side of the PIP command. Only a destination device can be specified in a delete command; it is assumed that the source and destination devices are the same device.

For example, the following command

DSK:/D=FILEA,FILEB,FILEC,MAC,*.REL<CR>

causes PIP to delete from the user's disk area files FILEA, FILEB, FILEC.MAC and all files having the extension .REL.

If a nonexistent file is specified in a delete command, PIP prints the error message

%filename.ext FILE WAS NOT FOUND

and continues to process deletions of the existing specified files. If an existing file is found to be protected it is skipped and the message

?filename.ext (2) PROTECTION FAILURE

is printed. If a user has the correct privileges he can delete files from other users' areas.

NOTE

An attempt to delete files from a DECtape that is write-locked results in the error message

DEVICE dev.name OPR operator
ACTION REQUESTED

being printed at the user's terminal. When the system operator has write-enabled the DECtape unit involved, he will start the requested action and cause the message

CONT BY OPER

to be printed at the user's terminal.

During the delete operation, PIP lists the names of the files deleted and the total number of blocks freed by the deletion. The names are typed before the files are actually deleted.

For example, assume that a file three blocks in length and named FILEA.MAC exists in the current UFD. The command for its deletion and the subsequent messages printed by PIP would appear as:

*DSK:/D=FILEA.MAC <CR>	(user command)
FILES DELETED:	(PIP response)
FILEA.MAC	(PIP response)
3 BLOCKS FREED	(PIP response)
*	

3.5.5 Z Switch, Zero Directory

The use of this switch causes PIP to zero out the directory of the destination's device; a source device does not have to be specified in the command. A Z-switch request is implemented before any other operation specified in the command string in which it occurs. Thus,

DTA2:CARDS/Z=CDR:<CR>

zeroes out the directory of DTA2 before transferring one file from CDR onto DTA2. The command

DTA2:/Z=<CR>

zeroes out the directory of DTA2.

If the destination device is the disk, an attempt is made to delete all the files whose names are found in the directory specified. If protection codes prohibit the deletion of some of the files, the request will terminate after as many files as possible have been deleted, and the message

?filename.ext(2)PROTECTION FAILURE

is printed. The user should then change the protection of the protected files and repeat his request if he wants all files deleted. For example, the command

DSK:FLOUT/Z=DTA2:CARY<CR>

zeroes out the directory of the user's disk area, transfers file CARY from DTA2 to the disk, and names the disk file FLOUT.

3.5.6 Q-Switch, Print Summary of PIP Functions

This switch causes PIP to print on a specified device the help file SYS:PIP.HLP. This file contains an alphabetical list of all PIP switches and functions. For example, the command

LPT:/Q=<CR>

causes the following summary to be listed on the line printer:

PIP	Switches (Alphabetic order) Summary
A	Line Blocking
B	Binary Processing (Mode)
C	Suppress Trailing Spaces, Convert Multiple Spaces to TABs
D	Delete File
E	Treat (Card) Columns 73-80 as Spaces

PIP Switches (Alphabetic order) Summary

F	List Disk or DTA Directory (Filenames and Ext. only)
G	Ignore I/O Errors
H	Image Binary Processing (Mode)
I	Image Processing (Mode)
J	Punch Cards in ASCII (Output Device must be CDP)
L	List Directory
M	See MTA Switches Below
N	Delete Sequence Numbers
O	Same as /S Switch, except increment is by 1
P	FORTRAN output Conversion assumed. Convert format control character for LPT listing. /B/P FORTRAN Binary
Q	Print (this) List of Switches and Meanings
R	Rename File
S	Resequence, or Add Sequence Number to File; increment is by 10
T	Suppress Trailing Spaces Only
U	Copy Block 0 (DTA)
V	Match parentheses (< >)
W	Convert TABs to Multiple Spaces
X	Copy Specified Files
*Y	RIM, DTA to PTP if source extension is RTB Destination format is RIM Loader, RIM 10B transfer word. If source extension is SAV destination format is as RTB – RIM 10B file only. If source extension is RMT destination format is RIM10.
Z	Zero Out Directory

MTA switches:

Enclose in parentheses ( ).

M followed by 8 means select 800 B.P.I. Density

5	556 B.P.I. Density
2	200 B.P.I. Density
E	Even Parity
A	Advance MTA1 File
D	Advance MTA1 Record
B	Backspace MTA1 File
P	Backsapce MTA1 Record
W	Rewind MTA or DTA
T	Skip to Logical EOT
U	Rewind and Unload MTA or DTA
F	Mark EOF

(M#NA), (M#NB), (M#ND), (M#NP) means advance or backspace MTAn files, or records.

*This is an optional switch obtained by setting RIMSW=1 at assembly time.

3.6 PERMITTED SWITCH COMBINATIONS

The combinations of PIP's standard and special option switches which are permitted in PIP commands are illustrated in the following matrix.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	NOTES
A		#	x	#	x	#	x	#	#	#		#	x	x	x		x	#	x			?		#		x	ASCII mode
B	#		#	#	#	#	x	#	#	#		#	x	#	#	*	x	#	#	#		#	#	x	#		binary mode
C	x	#		#	x	#	x	#	#	x			x	x	x		x	#	x	x		?	x	x	#	x	ASCII mode
D	#	#	#		#	#		#	#	#			#	#	#		#					#		*		#	Delete only
E	x	#	x	#		#	x	#	#	x			x	x	x		x	#	x					x	#	x	ASCII mode
F	#	#	#	#	#			#	#	#		#		#	#	#	#	#	#	#	#	#	#	#	#		List Directory only
G	x	x	x		x	x		x	x	x			x	x	x		x					x		x		x	Always legal
H	#	#	#	#	#	#	x		#	#			x	#	#	#							#	x		x	Binary mode
I	#	#	#	#	#	#	x	#		#			x	#	#	#		#	#	#			#	x		x	Binary mode
J	x	#	x	#	x	#	x	#	#				x	?	?			#			#		x	x	#	?	ASCII mode
K																											Unused
L	#		#			#																		#		x	List Directory only
M	x	x	x	#	x	#	x	x	x	x				x	x		x				#	x		x		?	Magnetic Tape only
N	x	#	x	#	x	#	x	#	#	?			x				x	#						x		x	ASCII mode
O	x	#	x	#	x	#	x	#		?			x	#				#	#					x		x	ASCII
P		*				#			#																	x	FORTRAN
Q	x	x	x	#	x	#	x						x	x	x			#				#		x		x	Prints file PIP.HLP
R	#	#	#			#			#	#					#		#	#				#		x		#	Rename only
S	x	#	x			#			#											x						x	ASCII mode
T		#	x			#		#											x							x	ASCII mode
U						#			#	#			#													x	DTA only
V	?	#	?	#		#	x						x				#	#								x	ASCII mode
W		#	x			#		#	#																	x	ASCII mode
X	x	x	x	*	x	#	x	x	x	x		#	x	x	x		x	x								x	ASCII or binary mode
Y	#	#	#		#	#																				x	Binary mode
Z	x	x	x	#	x		x	x	x	?		x	?	x	x	x	x	#	x	x	x	x	x	x	x		Output only

LEGEND:	Symbol	Meaning
	x	A permitted combination
	?	A permitted but unlikely combination
	#	Not permitted
	*	Special purpose combination
	Blank	Untested or unused combination

CHAPTER 4

SPECIAL PIP SWITCHES

This section contains descriptions of optional PIP functions used in magnetic tape, error recovery and card punch operations.

4.1 MAGNETIC TAPE SWITCHES

When magnetic tape is used in a file transfer, PIP can set the tape parity and density parameters and position the tape reels. In PIP commands, magnetic tape switches apply to only one particular magnetic tape unit or file specification.

The optional PIP magnetic tape (MTA) switches are written enclosed in parentheses; the letter M is used as the first character of all optional switches or series of switches (e.g., (MSW) or (MSW1SW2. .).

MTA switches must appear within the command file specifications of the particular file to which they refer. Thus, MTA switches refer to a particular device and, except for density and parity selections, to a particular file specification of that device.

4.1.1 Switches for Setting Density and Parity Parameters

The default Monitor density of 800 bits-per-inch (bpi) and odd parity are assumed unless either the Monitor SET DENSITY command was given or one of the following switches is included in the PIP command file specifications:

Switch	Meaning
(M8)	800 bpi density (default value)
(M5)	556 bpi density
(M2)	200 bpi density
(ME)	Even parity (odd parity is default)

The following command string causes PIP to transfer a file from MTA1 to MTA2 at 200 bpi, with even parity (and in ASCII line mode)

```
MTA2:(M2E)=MTA1:(ME2) <CR>
```

4.1.2 Switches for Positioning Magnetic Tape

The following switches are used in PIP command strings for magnetic tape handling:

Switch	Function Performed
(MA)	Advance tape reel one file.
(MB)	Backspace tape reel one file.
(MD)	Advance tape reel one record.
(MP)	Backspace tape reel one record.
(MW)	Rewind tape reel (works for DECtape also).
(MT)	Skip to logical End-of-Tape.
(MU)	Rewind and unload (works for DECtape also).
(MF)	Mark End-of-File.

In PIP MTA commands, the source device need not be given. For example, to rewind MTA1:, type

```
MTA1:(MW)=<CR>
```

If a source device is specified in the command string, information transfer will occur, except when PIP is requested to rewind and unload a magnetic tape.

Several magnetic tape functions may be specified in a single command string. Density or parity, when changed, will appear in the file specification. In the following example, density is set to 200 bpi, parity is even, the tape is to be rewound and the first, third, fourth and fifth files on that reel are to be printed on the line printer.

```
LPT:=MTA1:(M2EW),(MA),,<CR>
```

If multiple backspace, advance file or record movements are needed, the number of movements required is specified by #n (interpreted as decimal). All positioning switches are implemented before any related file transfers are made; thus MTA1:(M#3A)=PTR: will advance MTA1 by three files before transferring a paper tape file to it.

1. If a backspace file request (M#nB) is given, PIP backspaces over "n+1" files, then advances over one file – unless the tape is at Load Point. In this way the tape is always initially positioned at the beginning of a file. Thus, the command:

   ```
   MTA0:(MB)=<CR>
   ```

   will backspace MTA0 to the start of the previous file.

2. If the Load Point is reached before a backspace file or record request is completed, an error diagnostic will terminate the run and the following error message is printed

   ?LOAD POINT BEFORE END OF BACKSPACE REQUEST?

3. Only one MTA movement per file specification is allowed in a command string. Thus:

   MTA0:(MT#28)=. . . <CR>

   is illegal since it requests two distinct types of MTA movement.

4.1.2.1 Backspace to Start of Current File – The specification of 0 as the value of n in a multiple backspace command (e.g., M#0B) causes the tape to be backspaced to the start of the current file. The use of M#0B is not the same as MB; switch MB is equivalent to M#1B.

4.1.2.2 Advance to End of Current File – The specification of 0 as the value of n in a multiple advance command (e.g., M#0A) causes the tape to be moved to a point just before the EOF marker of the current file. The use of M#0A is not the same as MA; switch MA is equivalent to M#1A.

NOTE

The advance and backspace record requests are available as a convenience for the knowledgeable user, and should be approached with caution. Always remember that PIP typically has multiple input and output buffers and the physical position of the tape need not correspond to the physical position of the record currently being processed.

4.2 G-SWITCH, ERROR RECOVERY

If the error recovery switch /G is present in a command string, a specific set of I/O errors will be acknowledged by error messages. The I/O errors affected by the presence or absence of /G are listed in Section 5, paragraph 5.2, item 3 of the error messages, and are flagged by an asterisk (*). Processing will continue after the error message is printed as though no error had occurred. Thus, most I/O errors occurring within a file may be overridden. However, if the same error condition occurs in each buffer of the file, the error message is repeated for each buffer until either the end of file occurs or the error condition disappears. A disk directory is used as an input file if it is read to be either listed or searched and is obtained as a core image from the monitor; therefore, it is not subject to the input errors which may be diagnosed by PIP. However, I/O errors can occur for DECtape directories and are diagnosed at the monitor level when a directory is read or written. This occurs, typically, on a LOOKUP or RELEAS request. If the G-switch is not used, any I/O error will close the current file and, after printing a suitable message, will terminate the current request to PIP.

4.3 J-SWITCH, CARD PUNCH

The J-switch causes cards to be punched in ASCII mode. The output device specified by the command string must be the card punch (CDP).

CHAPTER 5

PIP ERROR REPORTING AND ERROR MESSAGES

This section describes the various types of error conditions and error messages that can occur during PIP operations.

Also described is the special treatment of recoverable error messages, which prevent the premature termination of a job running under the Batch Processor.

When an error message terminates a PIP run, both the input and output devices are released. This means that all files, fully or partly created, are available on the destination device.

NOTE

A question mark (?) preceding an error message indicates that the error is fatal (non-recoverable).

5.1 I/O ERROR MESSAGES

I/O error messages are opened with a description of the relevant device and file; for example,

1. INPUT DEVICE DTA3:FILE filename.ext . . .
2. OUTPUT DEVICE DTA3:FILE filename.ext . . .
3. disk directory read . . .

Device	Message
DTA,DSK,MTA	WRITE (LOCK) ERROR
*CDR	7-9 PUNCH MISSING
*OTHER	BINARY DATA INCOMPLETE
*ALL DEVICES	DEVICE ERROR
*ALL DEVICES	CHECKSUM OR PARITY ERROR
DTA	BLOCK OR BLOCK NUMBER TOO LARGE
*OTHER	INPUT BUFFER OVERFLOW
*MTA	PHYSICAL EOT

Thus, for the command DTA4:CON.REL=DTA3:CON.REL, if DTA4 is WRITE LOCKed, PIP prints the error message:

?OUTPUT DEVICE DTA4:FILE CON.REL WRITE(LOCK)ERROR

*Recoverable error if a G-switch is used; read paragraph 4.3 for a description of /G.

Other messages for devices are:

1. ?DEVICE dev DOES NOT EXIST (DEVCHR request)

2. ?DEVICE dev NOT AVAILABLE (INIT request)

5.2 FILE REFERENCE ERRORS

The following error messages can occur during a LOOKUP, RENAME or ENTER request on disk.

MESSAGE:? (FILENAME.EXT) THEN ONE OF THE FOLLOWING:

(0) FILE WAS NOT FOUND or (0) ILLEGAL FILE NAME
(used for enter errors only)
(1) NO DIRECTORY FOR PROJECT-PROGRAMMER NUMBER
(2) PROTECTION FAILURE
(3) FILE WAS BEING MODIFIED
(4) RENAME FILE NAME ALREADY EXISTS
(5) ILLEGAL SEQUENCE OF UUOS
(6) BAD UFD OR BAD RIB
(7) NOT A SAV FILE
(10) NOT ENOUGH CORE
(11) DEVICE NOT AVAILABLE
(12) NO SUCH DEVICE
(13) NOT TWO RELOC REG. CAPABILITY
(14) NO ROOM OR QUOTA EXCEEDED
(15) WRITE LOCK ERROR
(16) NOT ENOUGH MONITOR TABLE SPACE
(17) PARTIAL ALLOCATION ONLY
(20) BLOCK NOT FREE ON ALLOCATION
(21) CAN'T SUPERSEDE (ENTER) AN EXISTING DIRECTORY
(22) CAN'T DELETE (RENAME) A NON-EMPTY DIRECTORY
(23) SFD NOT FOUND
(24) SEARCH LIST EMPTY
(25) SFD NESTED TOO DEEPLY
(26) NO-CREATE ON FOR SPECIFIED SFD PATH

If the error code (V) is greater than 26(8), the error message:

?(V) LOOKUP,ENTER, OR RENAME ERROR

is printed.

Error values are used by the UUO's LOOKUP, ENTER and RENAME. Refer to the DECsystem-10 Monitor Calls Manual for complete descriptions of these UUO's.

The following error messages may be given on a reject to an ENTER request on DECtape.

1. The error message printed if there is no room for an entry in a DECtape directory is

   ?DIRECTORY FULL:

2. The error message printed if a zero filename is given for a DECtape output file is

   ?ILLEGAL FILE NAME:

The following message is given if a filename is not found in a directory search or disk or DECtape:

?NO FILE NAMED filename.ext

5.3 PIP COMMAND ERRORS

The following error messages are output by PIP on the detection of errors in the user command string:

1. ?PIP COMMAND ERROR

   Some of the possible causes of this type of error are:

   a. an illegal format for a command string,

   b. a nonexistent switch was requested,

   c. a filename other than * or *.* was given for a non-directory (source) device.

2. ?INCORRECT PROJECT-PROGRAMMER NUMBER:

   The project-programmer number must be in the form

   [number,number]

   where number is an octal number from 1 through 377777, a full path specification must be made if SFD's are involved.

3. ?SFD LIST TOO LONG:

   Too many SFD's were listed in the full directory path. A maximum of five levels (not including the UFD) is permitted in a directory path specification.

4. ?ILLEGAL PROTECTION:

   The protection number must be in the form <number>, where number is an octal number from 0 through 777.

5. ?NO BLOCK 0 COPY

   The /U switch was specified, but PIP was not assembled to allow this.

6. ?TOO MANY REQUESTS FOR . . . (magnetic tape)

   Conflicting density and/or parity requests were given.

5.4 Y-SWITCH ERRORS

The following error messages occur only when the Y-switch is included in the PIP command string:

1. ?DTA to PTP ONLY*

   Only DECtape input and paper tape output are permitted.

2. ?/Y SWITCH NOT AVAILABLE THIS ASSEMBLY:

   The /Y switch was specified, but PIP was not assembled to allow this.

3. FILE filename.ext ILLEGAL EXTENSION:

   The extensions of the filenames given must be .RMT, .RTB or .SAV.

4. Filename.ext ILLEGAL FORMAT:

   The reasons for getting the diagnostic ILLEGAL FORMAT are:

   a. a zero length file was found,

   b. the required job data information was not available,

   c. a block overlapped a previous block (RIM 10),

   d. an EOF was found when data was expected,

   e. a pointer word was expected but not found in the source file.

5.5 GENERAL ERROR MESSAGES

The following is a list of the PIP error messages which are not included in any of the preceding categories:

1. ?DISK OR DECTAPE INPUT REQUIRED:

   This message is printed when a non-directory source device is specified for a PIP function which requires a directory-type source device.

2. ?filename.ext ILLEGAL FILE NAME:

   This message is output if an attempt is made to ENTER without giving a filename.

3. Errors found during /X, /Z, /D, and /R operations result in error messages which pertain to the specific error found. Error messages for these operations are printed only if no other fatal error occurs before the command string is processed. If another error does occur, its diagnostic takes precedence over the diagnostics for the above switch functions.

4. ?4K NEEDED:

   4K not currently available but is needed (for non-reentrant disk system).

5. ?DECTAPE I/O ONLY:

   The I/O device for a block 0 copy (/U switch) must be a DECtape.

6. ?TERMINATE /X.MAX. OF 999 FILES PROCESSED:

   PIP, during a /X copy function from a non-directory device, has processed 999 files. This is the maximum number of files which such a /X request can handle.

7. ?TOO MANY INPUT DEVICES:

   This error is for the /D and /DX functions; only one input device is allowed when these switches are used. If more than one device is specified in a /D command and the first device given is DSK, the disk files are deleted when this diagnostic is given.

8. ?NO FILE NAMED PIP.HLP:

   The data file requested by a PIP Q-switch is not available on the system device.

9. ?LINE TOO LONG:

   During an ASCII mode file transfer a line containing more than 180 characters was detected. This occurs only when switches entailing line processing are given (i.e., /A or /S).

10. ?LOAD POINT BEFORE END OF BACKSPACE REQUEST:

    This diagnostic occurs only if either the MTA (M#nB) or (M#nP) switch is used. If the Load Point is sensed before the "n" backspace files or records function is completed, an error is assumed to have been made by the user.

5.6 TMPCOR ERROR MESSAGES

If the temporary storage facilities provided by the UUO TMPCOR are used or are attempted to be used during PIP operations, the following error messages can occur:

1. ?TMPCOR NOT AVAILABLE:

2. ?NOT ENOUGH ROOM IN TMPCOR:

3. ?COMMAND NOT YET SUPPORTED FOR TMPCOR:

   Wild cards, /X, and /R are not supported.

4. nn TMPCOR WORDS FREE

   Number of word locations free in the TMPCOR storage area.

Refer to the DECsystem-10 Monitor Calls Manual for a description of the UUO TMPCOR.

APPENDIX A

STANDARD FILENAME EXTENSIONS

Table A-1
Filename Extensions

FILENAME EXTENSION	TYPE OF FILE	MEANING
ABS	Object	Absolute (nonrelocatable) program.
AID	Source	Source file in AID language.
ALG	Source	Source file in ALGOL language.
ALP	ASCII	Printer forms alignment.
ATO	ASCII	OPSER automatic command file.
AWT	Binary	Data for automatic wire tester.
B10	Source	Source file in BLISS-10.
B11	Source	Source file in BLISS-11.
BAC	Object	Output from the BASIC Compiler.
BAK	Source	Backup file from TECO or LINED.
BAS	Source	Source file in BASIC language.
BCM	ASCII	Listing file created by FILCOM (binary compare).
BCP	Source	Source file in BCPL language.
BIN	Object	Binary file.
BLB	ASCII	Blurb file.
BLI	Source	Source file in BLISS language.

Table A-1 (Cont.)
Filename Extensions

FILENAME EXTENSION	TYPE OF FILE	MEANING
BUG	Object	Saved to show a program error.
BWR	ASCII	Beware file listing warnings about a file or program.
BNC	ASCII	BINCOM output.
CAL	Object	CAL data and program files.
CBL	Source	Source file in COBOL language.
CCL	ASCII	Alternate convention for command file (@ construction for programs other than COMPIL).
CCO	ASCII	Listing of modifications to non-resident software.
CDP	ASCII, Binary	Spooled output for card punch.
CFC	ASCII	Compressed file compare. Group of .SCM files combined with PIP.
CKP	Binary	Checkpoint core image file created by COBOL operating system.
CHN	Object	CHAIN file.
CMD	ASCII	Command file for indirect commands (@ construction for COMPIL).
CMP	ASCII	Complaint file by GRIPE.
COR	ASCII	Correction file for SOUP.
CRF	ASCII	CREF (cross-reference) input file.
CTL	ASCII	MP batch control file.
DAE	Binary	Default output for DAEMON-taken core dump.
DAT	ASCII, Binary	Data (FORTRAN) file.

Table A-1 (Cont.)
Filename Extensions

FILENAME EXTENSION	TYPE OF FILE	MEANING
DCR	Binary	Core image save (DCORE).
DCT	ASCII	Dictionary of words.
DDT	ASCII	Input file to FILDDT.
DIR	ASCII	Directory from FILE command or DIRECT program.
DMP	PDP-6	PDP-6 format for a file created by a SAVE command.
DOC	ASCII	Listing of modifications to the most recent version of the software.
DRW	Binary	Drawing for VB10C drawing system.
DSE	ASCII	Directory sorted by extension.
DSF	ASCII	Directory sorted by filename.
ERR	ASCII	Error message file.
F4	Source	Source file in FORTRAN IV language.
FAI	Source	Source file in FAIL language.
FCL	Source	Source file in FOCAL language.
FFS	ASCII	Fast FORTRAN stream.
FLO	ASCII	English language flowchart.
FOR	Source	Source file in FORTRAN 10 language.
FRM	ASCII	Form.
FTP	Source	FORTRAN test programs.
FUD	ASCII	FUDGE2 listing output.
GND	ASCII	List of ground pins for automatic wire wrap.

Table A-1 (Cont.)
Filename Extensions

FILENAME EXTENSION	TYPE OF FILE	MEANING
HGH	Object	Nonsharable high segment of a two-segment program.
HLP	ASCII	Help files containing switch explanations, etc.
IDA	ASCII, Binary	COBOL ISAM data file.
IDX	ASCII, SIXBIT	Index file of a COBOL ISAM file.
INI	ASCII Binary	Initialization file.
LAP	ASCII	Output from the LISP compiler.
LIB	ASCII	COBOL source library.
LOG	ASCII	MP batch log file.
LOW	Object	Low segment of a two-segment program.
LPT	ASCII	LISTING DATA.
LSD	ASCII	Default output for DUMP program.
LSP	Source	Source file in LISP language.
LSQ	ASCII	Queue listing.
LST	ASCII	Listing data.
MAC	Source	Source file in MACRO language.
MAN	ASCII	Manual (documentation) file.
MAP	ASCII	Loader map file.
MEM	ASCII	Memorandum file, typically output from RUN-OFF.

Table A-1 (Cont.)
Filename Extensions

FILENAME EXTENSION	TYPE OF FILE	MEANING
MID	Source	Source file in MIDAS (MIT Assembler) language.
MIM	Binary	Snapshot of MIMIC simulator.
MSB	Object	Music compiler binary output.
MUS	Source	Music compiler input.
N	Source	Source file in NELIAC language.
NEW	All	New version of a program or file.
OBJ	Object	PDP-11 relocatable binary file.
OLD	Source	Backup source program.
OPR	ASCII	Installation and assembly instructions.
OVR	Object	COBOL overlay file.
P11	Source	Source file for PDP-11.
PAK	Source	Files compressed by PACK.TEC to save disk space.
PAL	Source	Source file in PAL 10 (PDP-8 assembler).
PBT	ASCII	P-batch control file.
PL1	Source	Source file in PL1 language.
PLG	ASCII	P-batch log file.
PLM	ASCII	Program Logic Manual.
PLO	Binary	Compressed plot output.
PLT	ASCII	Spooled output for plotter.
PPL	Source	Source file in PPL language.
PRO	Object	Program (save file).

Table A-1 (Cont.)
Filename Extensions

FILENAME EXTENSION	TYPE OF FILE	MEANING
PTP	ASCII, Binary	Spooled output for paper-tape punch.
Qxx	ASCII	Edit backup file, like .BAK (all xx).
QUC	Binary	Queue change request file.
QUD	ASCII Binary	Queued data file.
QUE	Binary	Queue request file.
QUF	Binary	Master queue and request file.
REL	Object	Relocatable binary file.
RIM	Object	RIM loader file.
RMT	Object	Read-In mode (RIM) format file (PIP).
RNC	ASCII	RUNOFF input for producing a .CCO file.
RND	ASCII	RUNOFF input for producing a .DOC file.
RNH	ASCII	RUNOFF input for producing an .HLP file.
RNL	ASCII	RUNOFF input for producing a .PLM file.
RNO	ASCII	Programming specifications in RUNOFF input.
RNP	ASCII	RUNOFF input for producing a .OPR file.
RSP	ASCII	Script response time log file.
RSX	All	Files for RSX-11D.
RTB	Object	Read-In mode (RIM10B) format file (PIP).
SAI	Source	Source file in SAIL language.

Table A-1 (Cont.)
Filename Extensions

FILENAME EXTENSION	TYPE OF FILE	MEANING
SAV	Object	Low segment from a one-segment program.
SCD	ASCII	Differences in directory.
SCM	ASCII	Filcom output.
SCP	ASCII	SCRIPT control file.
SEQ	ASCII	Sequential COBOL data file, input to ISAM program.
SFD	Binary	Sub-file directory (restricted usage).
SHR	Object	Sharable high segment file of a two-segment program.
SMP	Source	Source file in SIMPLE language.
SNO	Source	Source file in SNOBOL language.
SNP	ASCII	Snapshot of disk by DSKLST.
SPT	ASCII	SPRINT - created files.
SRC	ASCII	SRCCOM output.
SVE	Object	.SAVed file from a single user Monitor.
SYM	Binary	LINK-10 symbol file.
SYS	Binary	Special System files.
TEC	ASCII	TECO macro.
TEM	ASCII, Binary	Temporary files.
TMP	ASCII, Binary	Temporary files.
TPC	ASCII	TYPESET input for producing a .CCO file.
TPD	ASCII	TYPESET input for producing a .DOC file.

Table A-1 (Cont.)
Filename Extensions

FILENAME EXTENSION	TYPE OF FILE	MEANING
TPH	ASCII	TYPESET input for producing an .HLP file.
TPL	ASCII	TYPESET input for producing a .PLM file.
TPO	ASCII	Programming specification in TYPESET input.
TPP	ASCII	TYPESET input for producing an .OPR file.
TST	All	Test data.
TXT	ASCII	Text file.
UFD	Binary	User file directory (restricted usage).
UPD	ASCII	Updates flagged in margin (SRCCOM).
VMX	Object	Expanded save file starting at a location greater than zero and used as a special support program for virtual memory.
WCH	ASCII	SCRIPT Monitor (WATCH) file.
WRL	ASCII	Wire list.
XOR	Binary	Module data for XOR tester.
XPN	Object	Expanded save file (FILEX).

INDEX

decsystem10

GETTING STARTED WITH RUNOFF

Text Formatting Program

Order No. DEC-10-URUNA-A-D

digital equipment corporation · maynard. massachusetts

1st Edition February 1975

The HOW TO OBTAIN SOFTWARE INFORMATION page, located at the back of this document, explains the various services available to DIGITAL software users.

The postage prepaid READER'S COMMENT form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

PREFACE

This manual is for the person who knows how to create a document with a text editor or Batch system but knows nothing about the RUNOFF program. The reader need not have any programming or mathematical skills. The descriptions of the most commonly used commands provide a tutorial on how to format documents the easiest and most efficient way. The entire list of commands and their meanings can be found in the Appendix.

This document reflects the software as of version 10.

CONTENTS

GLOSSARY

ARGUMENT	A variable whose value determines the value of a function.
ASCII	American Standard Code for Information Interchange. A 7-bit code in which textual information is recorded. The ASCII code can represent 128 distinct characters. These characters are the upper and lower case letters, numbers, common punctuation marks, and special control characters.
BUFFER	A special register or a designated area of internal storage.
CHARACTER	One symbol of a set of elementary symbols such as those corresponding to the keys on a typewriter. The symbols usually include the decimal digits 0 through 9, a space, the letters A through Z, operation symbols, and any other special symbols which a computer reads, stores, or writes.
DEFAULT	The value, name, or argument used unless you specify an alternative to override it.
FORMAT	The arrangement and organization of your document.
INDEX BUFFER	The internal storage area for words indexed during input.
JUSTIFICATION	The adjusting of spaces between words to align margins.
MODE	The current style or method of output.
n	A number which you must supply. It can contain as many digits as necessary.
QUEUE	A list of items waiting to be scheduled or processed according to system, operator, or user-assigned priorities.
RAGGED RIGHT	An uneven right margin.
SOURCE FILE or SOURCE TEXT	The document you input intermingled with RUNOFF commands.
TAB STOPS	Tabbing on a terminal is like tabbing on a typewriter. When you press the tab button on a terminal, you move the printing element 8 spaces to the right. The next character you type appears on the 9th space. If you tab again, the element moves 7 more spaces to the right, placing the next character you type on the 17th space. Every time you tab you move the element up to 8 spaces or columns to the right, depending on how many characters you have typed.

CONVENTIONS USED IN THIS MANUAL

↵	This indicates a carriage return.
text	You must supply a word or words.
(text)	You have the option of supplying a word or words but it is not necessary. You do not type the parentheses.

1.0 INTRODUCTION

RUNOFF is a DECsystem-10 program that enables you to prepare documents easily in conjunction with a text editor or Batch system. By inserting RUNOFF commands and special characters with your text, you can format your material with a minimum of effort.

You can use any editor you are familiar with such as TECO or LINED to input your documents. During input you need not worry about spacing between words, justifying lines, case shifting, page numbering, and other formatting considerations. RUNOFF does all these things for you. Because you add certain commands with your text, RUNOFF can take your file and reproduce it according to your specifications.

If you must make changes to your file, you do so with the text editor. After you make your corrections, you run your file through RUNOFF, and your document is again in the proper format. Once your document is ready, you can reproduce it either

1. Into another file,
2. On your terminal, or
3. On a line-printer.

By using RUNOFF you can update your documentation and make it look presentable without extensive retyping.

2.0 PRODUCING A SOURCE FILE

Your source file is the text you input plus the formatting and mode instructions you add for RUNOFF. The first thing you must do to produce a source file is to locate a terminal and log-in. Next create a file as you normally do by typing the text with a text editor. The difference is that while you are inputting your text, you also include special RUNOFF commands and special characters to be interpreted by RUNOFF during its execution.

As described later in section 3.0, all RUNOFF commands begin with a period (.) in column 1. Therefore, all of the material you type into your source file is taken to be source text by RUNOFF except for those words beginning with a period. These commands do not appear in your output. Because you can set left and right margins with RUNOFF commands, you can type your text freely without worrying about the width of the lines or the spacing between words. (However, you must type at least one space between words to indicate the end of a word.)

RUNOFF fills and justifies the text as it is processed. The FILLing is accomplished by adding successive words from the source text until the adding of one more word will exceed the right margin. RUNOFF stops before putting that last word in and JUSTIFIes the line by increasing the space between words until the last word in the line exactly meets the right margin.

2.1 Setting the Format

Certain modes and formatting instructions are already set up by RUNOFF before you even begin your input. These defaults are

1. Page numbers on every page except the first,
2. Spacing 1 between lines,
3. Fill and justify,
4. Tab stops 9, 17, 25, 33, 41, 49, 57, 65,
5. Left margin 0,
6. Right margin 60,
7. Page size - Width 60 characters, Length 58 lines.

However, there are RUNOFF commands that you must input at the beginning of your source file to set other format and mode operations. You decide how you want your document to look. Then, if you do not want the standard left and right margins of 0 and 60 and the standard page size 58/60, you begin your file by typing commands for

1. Left and right margins, and
2. Width and length of the page

NOTE

If your left and right margins are the same as the page size, you need only specify the page size.

You can also add various optional commands at the beginning or any other point of your source file to produce the following:

1. Title on every page,
2. Subtitle on every page,
3. Automatic paragraphing,
4. Ragged right,
5. Upper/lower case,
6. Change tab stops,
7. No page numbers,
8. Enable Capitalize Flag,
9. Enable Index Flag, and
10. Change horizontal spacing between sentences.

RUNOFF initially outputs in the same case as the input. If you only have an upper case terminal and you want your document to output in upper/lower case, you must set the mode at the very beginning of your source file to lower case. You do this by typing the command .LOWER CASE (see sections 3.4 and 4.1.4).

Ordinarily, RUNOFF treats spaces and carriage returns only as word separators. However, if you want to have a ragged right instead of an even right margin, you type the command .NOFILL at the beginning of the text. Then a carriage return will BREAK the line instead of just acting as a word separator.

If you are inputting a table, you must remember to type .

.NOFILL ↵

before you type your table, and then you must type

.FILL ↵

after your table. If you do not do this, your table will not appear in column form after you run your file through RUNOFF.

Because you set the format in advance, RUNOFF calls for new pages when necessary. You can also explicitly call for a page advance where desired by typing the .PAGE command.

NOTE

You can specify the normal settings with the .STANDARD n command.

Refer to section 3.0 for a full explanation of the commands discussed in this section.

2.2 Typing the Text

While you are inputting the source text, you have the option of inserting various RUNOFF commands. You type the command immediately before the text you want to format. Then you type one semicolon and then the text. For instance, if you want to center the word "MEMO" on a page, you type

.CENTER;MEMO ↵

The word "MEMO" will appear like this

MEMO

after you run your file through RUNOFF.

3.0 COMMANDS

All RUNOFF commands begin with a period (.). You can abbreviate all commands according to the standard abbreviations listed in this section. You can also abbreviate a long command by leaving off letters at the end of the command making sure this command remains unique from the others. However, you should not make extensive use of your own abbreviations, since they are not guaranteed unique for future versions of RUNOFF.

Some commands require arguments of one or more decimal numbers. Other commands require text arguments. In both instances, you type the command, one space, and then the argument. When you have several commands to be input, one after another, you can list all on one line separating them from each other with a period (command indicator). For example,

.LOWER CASE.NONUMBER.FLAG CAPITALIZE.FLAG INDEX ↵

However, if the first command has a comment or takes text as its argument, you cannot use the period as a separator. You can use a semicolon to separate multiple commands and comments on the same line. Comments must be preceded by an exclamation point (!).

.LEFT MARGIN 10 !comment; .TAB STOPS 15, 20, 30 !comment

In commands which take text strings as text, a period (.) and an exclamation point (!) are considered part of the text. You can only terminate these commands by typing a semicolon (;) or by typing a line terminator such as a carriage return or line feed.

NOTE

The only commands which cannot have another command on the same line are .TITLE, .SUBTITLE, and .INDEX because all characters are made part of the text.

While you are inputting your text, do not type a command in the middle of a line. You must start a new line placing the command at the beginning and the period in column one.

For example,

"This is the last sentence in my paragraph." ↵
.SKIP 2 ↵
"This is the first sentence in the next paragraph." ↵

This input skips two lines before starting the next paragraph.

The following commands are listed in four categories:

1. Text Formatting,
2. Page Formatting,
3. Mode Setting, and
4. Parameter Setting.

Standard abbreviations are given with the commands.

3.1 Text Formatting

.BREAK

.BREAK or .BR

causes the current line to be output with no justification and places the next word of the source text at the beginning of the next line.

.INDENT

.INDENT n or .I n

causes a BREAK and sets the next line to begin n spaces to the right of the left margin. The n can be negative. This allows you to begin a line to the left of the left margin. However, you cannot specify n to the left of 0.

.PARAGRAPH

.PARAGRAPH n, v, t or .P n, v, t

causes a BREAK and formats the output paragraphs. n is optional and, if present, sets the number of spaces the paragraph is to be indented. The default for n is 5. (n can also have a negative value.) v is the vertical spacing between paragraphs. v can be from 1 to 5. (1 is single spacing, 2 is double spacing, etc.) t causes an automatic .TEST PAGE. (See the .TEST PAGE command.)

.FIGURE

.FIGURE n or .FG n

leaves n lines blank to make room for a figure or diagram. If fewer than n lines remain on the current page, text continues to fill this page. Then the page is advanced and n blank lines are left at the top of the new page.

.SKIP

.SKIP n or .S n

causes a BREAK after which n is multiplied by the number of spaces between lines. The result is the number of lines skipped. Output is advanced to the top of the next page if there is no room on the current page. The n can also have a negative value. Thus, a final footnote can be set by a command such as .SKIP -5.

.BLANK

.BLANK n or .B n

causes the current line to be output with no justification, skips n line spaces, and then starts output of the source text at this point. n can be negative to move the line to n lines from the end of the page.

.CENTER

.CENTER n;text or .C n;text (.CENTRE n;text)

causes a BREAK and centers the following text in the source file. The centering is over column n/2 independent of the setting of the left and right margins. If n is not given, it is assumed to be the page width.

.FOOTNOTE

.FOOTNOTE n or .FN n

saves n lines at the bottom of the current page for a footnote. The n you specify with this command is multiplied by the number of spaces you previously set with the .SPACING command. If insufficient room remains on the current page, space is allocated at the bottom of the following page. You type the text of the footnote on the line following the command.

Indentation, case lock, justify, margins, spacing, and fill are preserved around footnotes. You can include commands within your footnote; however, commands which affect page formatting are illegal in a footnote. Tab stops are illegal because they are not preserved. A footnote within a footnote is also illegal.

The actual space taken by a footnote can be more or less than what you specify by n. You can adjust n after examining a first draft printout.

You must remember to end your footnote by typing an .END FOOTNOTE command.

.END FOOTNOTE

.END FOOTNOTE

terminates the footnote and signals the return to preserved formatting instructions.

.NOTE

.NOTE (text) or .NT (text)

starts an indented note. This command .BLANKs 2, reduces both margins by 15, centers the text (if no text is given, it centers the word "NOTE"), and then .BLANKs 1. At this point you type the text of the note. (You do not type the parentheses.)

.END NOTE

.END NOTE or .EN

terminates the .NOTE command, .BLANKs 2, and reverts the margins and spacing modes to their settings before the last .NOTE command.

.LIST

.LIST n or .LS n

starts an indented list with n spacing, moves the left margin 9 spaces to the right for the first .LIST command, and 4 more spaces to the right for each subsequent nested .LIST. The normal FILL and JUSTIFY modes remain in effect; therefore, you must disengage them just after the .LS command if you want a ragged right.

.LIST ELEMENT

.LIST ELEMENT; text or .LE; text

starts an item in the list, used in conjunction with the .LIST command. The elements are numbered sequentially and the number is given a negative indent so the list lines up. The number is followed by a period and two spaces (.##) so the indent will be –4. The list elements are separated by the standard paragraph spacing and .TEST PAGE. If you want to type the text on the same line as the command you must separate them with a semicolon. (See section 6.0 for an example of this command.)

.END LIST

.END LIST or .ELS

terminates the .LIST command and returns to settings before the last .LIST command.

3.2 Page Formatting

.CHAPTER

.CHAPTER text or .CH text

starts a new chapter using the text you input as the title of the chapter. This command acts as if you typed in
.BREAK.PAGE.BLANK 12;.CENTER "CHAPTER n"
The n is incremented by 1 automatically. After the "CHAPTER n" is typed on the page,
.BLANK 2; .CENTER;text; .BLANK 3
occurs. This command then resets the case, margins, spacing, and justify/fill modes. It also clears any subtitles and sets the chapter name as the title.

.HEADER LEVEL

.HEADER LEVEL n text or .HL n text

starts a section at the level specified and takes the following text as the header. n can be in the range from 1 to 5. The sections are incremented by 1 automatically, and the number is output in the form i.j.k.l.m. If this is a chapter oriented document, the i is the chapter n number; otherwise, it is the number of the .HL 1 level.

This command acts as a
.BREAK.TEST PAGE 9; .BLANK 3
followed by the section number, two spaces, and the section name. HEADER LEVELS 1 and 2 end with a .BREAK. HEADER LEVELS 3, 4, and 5 end with a space dash space combination (#-#).

.NUMBER

.NUMBER n or .NM n

starts page numbering. This is the default so there is no reason for you to issue this command unless you disengage page numbers. If you want to resume at a certain page, you specify n.

.NONUMBER

.NONUMBER or .NNM

disengages page numbering; however, pages continue to be counted so the normal page number can appear if you re-enable page numbers with the .NUMBER command.

.TITLE

.TITLE text or .T text

takes the remaining text as the title and outputs it on every page at line 0. The default is no title. If you want a title you must type this command into your source file.

.SUBTITLE

.SUBTITLE text or .ST text

takes the remaining text as the subtitle and outputs it on every page. It appears directly under the title. The subtitle is not indented but you can achieve the same results by typing leading spaces.

.DO INDEX

.DO INDEX (text) or .DX (text)

forces a new page, centers the text, if given, otherwise it centers the word "INDEX". (A period (.) terminates this command unless the period is quoted.) (You do not type the parentheses.)

This command causes a BREAK and then prints the entire contents of the index buffer. Entries are printed in alphabetic order and are set against the left margin. Regular line spacing is used, except that a blank line is left between entries of different first letters. The page number of each entry is placed on the same line as the entry and in the middle of the page. Additional page numbers for multiple entries follow, separated by commas. The index buffer is left empty.

.PAGE

.PAGE or .PG

causes a BREAK and an advance to a new page. If the current page is empty, this command does not advance the page. Just like the automatic page advance, this command adds the title (if given) and page numbers on every page.

.TEST PAGE

.TEST PAGE n or .TP n

causes a BREAK followed by a conditional page advance. It skips to the next page if fewer than n lines are left on the page. This capability is to ensure that the following n lines are all output on the same page. This command has the form t as an optional argument to the .PARAGRAPH command.

3.3 Mode Setting

.AUTOPARAGRAPH

.AUTOPARAGRAPH or .AP

causes any blank line or any line starting with a space to be considered as the start of a new paragraph. This command allows normally typed text to be justified without special commands. It does not cause a paragraph if blank lines are followed by a command.

.NOAUTOPARAGRAPH

.NOAUTOPARAGRAPH or .NAP

disengages the .AUTOPARAGRAPH mode.

.FILL

.FILL or .F

adds successive words from the source text until the adding of one more word will exceed the right margin. It stops before putting that last word in. FILL is the default mode. (.FILL also restores .JUSTIFY.)

.NOFILL

.NOFILL or .NF

disengages the .FILL and .JUSTIFY modes. You use this command when you are typing a table.

.JUSTIFY

.JUSTIFY or .J

increases spaces between words until the last word exactly meets the right margin. This is the default mode.

.NOJUSTIFY

.NOJUSTIFY or .NJ

stops justification of lines to make a ragged right margin.

.UPPER CASE

.UPPER CASE or .UC

sets the output mode to upper case. This command acts the same as typing two circumflexes (^^). This mode is the default. There is no need for you to type in this command unless you previously altered the mode to lower case.

.LOWER CASE

.LOWER CASE or .LC

sets the mode of typeout to lower case. This command acts the same as typing two backslashes (\\).

.FLAG CAPITALIZE

.FLAG CAPITALIZE

enables the less-than (<) character to capitalize the entire word it precedes. It then returns the file to the current case mode. This special character is initially off. You must type this command at the very beginning of your source text if you want to enable this character. Typing a space or another less-than (<) returns the file to the current case lock.

.FLAG INDEX

.FLAG INDEX

enables the greater-than (>) character to place the word it precedes into an index buffer. You must type this command at the beginning of your text if you want to enable this character. Then when you type the .DO INDEX before you close your source file, RUNOFF outputs a ready-made index for your document.

3.4 Parameter Setting

.LEFT MARGIN

.LEFT MARGIN n or .LM n

sets the left margin to n. The n must be less than the right margin but not less than 0. The default setting is 0.

.RIGHT MARGIN

.RIGHT MARGIN n or .RM n

sets the right margin n. The n must be greater than the left margin. The default setting is 60.

.PAPER SIZE

.PAPER SIZE n,m or .PAGE SIZE n,m or .PS n,m

sets the size of the page n lines by m columns. The default setting is 58, 60.

.SPACING

.SPACING n or .SP n

sets the number of spaces you want between lines. The n can be from 1 to 5. The default setting is 1. .SPACING 1 is like single spacing on a typewriter and .SPACING 2 is like double spacing on a typewriter. .SPACING 2 puts one blank line between sentences.

.STANDARD

.STANDARD n or .SD n

returns all settings to the standard defaults. If you specify .STANDARD 60, you reset margins to LM 0 RM 60, page size to 60, 58, .SPACING 1 between lines, and paragraph indent to 0. .STANDARD 70 sets your right margin to 70 and your page size to 70, 64. (Page length is directly related to the page width.) This command is useful if you change your settings, and you want to reinitialize the standard block format. (It does not alter tabs or paragraph numbering.)

.TAB STOPS

.TAB STOPS n, n, . . . or .TS n, n, . . .

sets tabs. The n must be greater than 0 and listed in ascending order. Tabs can only be used in lines that are unjustified and unfilled. If tabs already exist, the issuing of another .TAB STOPS command clears all previous tabs before setting the new ones. The default tabs are set at eight-column intervals just like the hardware standard on your terminal. These tabs are at columns 9, 17, 25, 33, 41, 49, 57, 65.

If you forget to disable FILL and JUSTIFY, your tabs will not work in your output file.

4.0 SPECIAL CHARACTERS

While inputting your text, you have the option of including special characters to alter the case and mode operations. You type these characters immediately before the word or group of words you want to arrange. Just like RUNOFF commands, these special characters do not appear in your output after running RUNOFF. Special text characters include

1.	Underscore	(_)	takes next character as text.
2.	Circumflex	(^)	upper case next character.
3.	Back-slash	(\)	lower case next character.
4.	Number sign	(#)	expandable space.
5.	Ampersand	(&)	underline next character.
6.	Less-than	(<)	capitalize the following word.
7.	Greater-than	(>)	index the following word.
8.	Exclamation point	(!)	end footnote or begin a comment.
9.	Period	(.)	indicates a RUNOFF command.
10.	Semicolon	(;)	separates multiple commands.

4.1 Case Shifting

Because some terminals only type in upper case mode, RUNOFF has special characters you can input to change the mode to lower case and back to upper case. These characters are echoed on your terminal during input so you can see exactly what you are doing to your source file, but they do not appear in your output file after executing RUNOFF.

4.1.1 Upper Case – The upper case character is the circumflex (^). If you want a particular letter of a word to be capitalized, you type a circumflex (^) immediately before the letter. After executing RUNOFF, the letter will be output in upper case, and the circumflex will not be output. For example, if you want the first letter of the word "computer" to be capitalized, you type

```
^ computer
```

This word will be output as

```
Computer
```

You type a circumflex (^) for the same reason you use the shift key on a typewriter.

4.1.2 Upper Case Lock – Initially RUNOFF outputs in upper case mode when you use an upper case only terminal for your input. But if you change the mode to lower case (see 4.1.3) and you want to return to upper case, type two circumflexes (^^). Two circumflexes (^^) lock the mode in upper case just like the shift lock on a typewriter. Everything you input after you type ^^ is output in the same case as the input.

4.1.3 Lower Case – The lower case character is the back-slash (\). If you type a back-slash (\) immediately before a letter, it outputs in lower case mode. For example, the word "computer" typed

```
\COM\PU\TER
```

outputs as

```
cOMpUtER
```

4.1.4 Lower Case Lock – Some terminals input in upper case only. In this instance, begin your file with two back-slashes (\\). (You can also use the .LOWER CASE command.) You can then indicate the words you want capitalized by typing a circumflex (^). The following example shows the use of the case control characters. (The current mode is lower case.)

```
^HERE IS A ^SAMPLE ^SENTENCE IN ^^UPPER CASE
\\AND LOWER CASE.
```

After running this sentence through RUNOFF, it looks like this.

```
Here is a Sample Sentence in UPPER CASE
and lower case.
```

4.2 Underlining

The underlining character in RUNOFF is the ampersand (&). If you want a character to be underlined in your output, you type an ampersand (&) immediately before the letter. For example, (mode is lower case)

```
&T&Y&P&E
```

outputs as

type

If you want to lock the underline character on so that you can underline an entire sentence you type circumflex ampersand (^&). This combination of characters underlines all characters except space. If you want to disengage this mode, you type backslash ampersand (\&).

4.3 Quoted Space

If you want to include a space without having the FILL and JUSTIFY expand it to the right margin, type a number sign (#) for every space you want. This number sign (#) is not treated as just a word separator. It explicitly calls for a space right where you request it.

4.4 The Capitalize Flag

The less-than character (<) typed before a word capitalizes the entire word and then returns the file to the current mode. Less-than (<) has the same effect as typing two circumflexes (^^) before a word and then two backslashes (\\) after the word. However using this character prevents you from forgetting to return to the current mode.

You must remember that, unlike the preceding characters, the less-than (<) is not initially turned on. If you do not enable it by typing the command .FLAG CAPITALIZE at the beginning of your source file (see section 3.3) you will only succeed in typing out a less-than (<) at the beginning of your source file if you plan to use it in your source text.

You can terminate this character by typing a space after the word or by typing another less-than (<).

```
<dec<system-10
```

outputs as

```
DECsystem-10
```

4.5 The Index Flag

The character greater-than (>) typed before a word automatically puts that word in an index buffer. You end the index entry with the first space, a new line, or the second occurrence of the index flag character >. You must type this character immediately before the word you want indexed (except when including the capitalize flag). By typing <> before a word, you can place a fully capitalized word into an index. Otherwise, the default index case mode prevails. This mode automatically capitalizes the first letter of the index entry, for example,

```
for >example this >par-3-Foo bar
```

causes index entries

"Example" and "Par-3-Foo."

This special character is initially off just like the capitalize flag <. Therefore, you must enable it at the beginning of your source file (see section 3.1). If you do not enable this character, your output file will contain greater-than (>) signs, and you will not have an index to output with the .DO INDEX command (see section 3.2).

4.6 Printing Special Characters

Sometimes you may want special characters to be output in your final document just as they appear in your source file. If your document discusses a circumflex (^), you may want your final output to print a ^ without changing anything to upper case. The special character (underscore (_) typed immediately before the character allows you to output these special characters just as they are in the source text without transmitting formatting instructions. For example,

_^ outputs as ^

If you want the underscore character to output in your document you also quote it by typing another underscore immediately before it in your source text.

5.0 HOW TO RUN RUNOFF

Now that you have completed your source text, you are ready to use the RUNOFF program. The first thing you do is to call RUNOFF by typing

.R RUNOFF ↵

RUNOFF responds with an asterisk (*) to tell you it is ready for your input. The next thing you do is type your filename. You can add the extension .RNO to your filename if you want to. Do not worry if you leave it off.

*filename.RNO ↵

Your file is processed through RUNOFF, and the system responds with the filename, the number of pages contained in the resulting output file, any errors which have occurred in processing, and the input pages and output pages where your errors occurred, if any. For example, with a filename of TEST your dialogue looks like

```
.R RUNOFF ↵
*TEST.RNO ↵
TEST          2 PAGES
*
```

This response tells you that you have no errors and your output is two pages long. The asterisk (*) indicates that RUNOFF is waiting for another file to process. At this point you type a CONTROL-C (^C) to exit back to monitor mode. You now have an output file in your area with the extension .MEM. Now you can see what your output looks like by

1. Having it typeout on your terminal, or
2. Queue it to a line-printer.

The output file for the previous example is TEST.MEM.

If you have made errors, your dialogue with RUNOFF may look like this

```
.R RUNOFF
*TEST.RNO
TEST
%RNFJEC JUNK AT END OF COMMAND: ". "
            ON OUTPUT PAGE 1; ON INPUT LINE 8 of PAGE 1
   TOTAL 2 PAGES
*
```

You correct your errors with your text editor and then run your file through RUNOFF again. You need not retype the entire document to change an error.

For a complete list of error messages and explanations, turn to section 7.0

6.0 EXAMPLES

This section contains various examples of procedures you may like to use with RUNOFF.

Here is the source file for the introduction of this manual, section 1.0.

EXAMPLE 1

```
.LOWER CASE,FLAG CAPITALIZE
1.0##<INTRODUCTION
.SKIP 2
<RUNOFF IS A <DEC<SYSTEM-10 PROGRAM THAT ENABLES YOU TO PREPARE
DOCUMENTS EASILY IN CONJUNCTION WITH A TEXT EDITOR OR ^BATCH
SYSTEM.##^BY INSERTING <RUNOFF COMMANDS AND SPECIAL CHARACTERS
WITH YOUR TEXT, YOU CAN FORMAT YOUR MATERIAL WITH A MINIMUM OF
EFFORT.
.SKIP 2
^YOU CAN USE ANY EDITOR YOU ARE FAMILIAR WITH SUCH AS <TECO OR
<LINED TO INPUT YOUR DOCUMENTS.##^DURING INPUT YOU NEED NOT WORRY
ABOUT SPACING BETWEEN WORDS, JUSTIFYING LINES, CASE SHIFTING, PAGE
NUMBERING, AND OTHER FORMATTING CONSIDERATIONS.##<RUNOFF DOES ALL
THESE THINGS FOR YOU.##^BECAUSE YOU ADD CERTAIN COMMANDS WITH YOUR
TEXT, <RUNOFF CAN TAKE YOUR FILE AND REPRODUCE IT ACCORDING
TO YOUR SPECIFICATIONS.
.SKIP 2
^IF YOU MUST MAKE CHANGES TO YOUR FILE, YOU DO SO WITH THE TEXT
EDITOR.##^AFTER YOU MAKE YOUR CORRECTIONS, YOU RUN YOUR FILE
THROUGH <RUNOFF, AND YOUR DOCUMENT IS AGAIN IN THE PROPER FORMAT.
^ONCE YOUR DOCUMENT IS READY, YOU CAN REPRODUCE IT EITHER
.SKIP 2;.NOFILL
        1.  ^INTO ANOTHER FILE,
        2.  ^ON YOUR TERMINAL, OR
        3.  ^ON A LINE-PRINTER.
.SKIP 2;.FILL
^BY USING <RUNOFF YOU CAN UPDATE YOUR DOCUMENTATION AND MAKE IT
LOOK PRESENTABLE WITHOUT EXTENSIVE RETYPING.
```

This is how it looks after running the file through RUNOFF.

```
1.0  INTRODUCTION


RUNOFF is a DECsystem-10 program that enables you to prepare
documents  easily in conjunction with a text editor or Batch
system.  By inserting RUNOFF commands and special characters
with  your text, you can format your material with a minimum
of effort.


You can use any editor you are familiar with such as TECO or
LINED  to  input  your documents.  During input you need not
worry about spacing between words,  justifying  lines,  case
shifting,    page    numbering,    and    other   formatting
considerations.  RUNOFF  does  all  these  things  for
you.  Because  you  add  certain  commands  with  your text,
RUNOFF can take your file and reproduce it according to your
specifications.


If you must make changes to your file, you do  so  with  the
text  editor.  After you make your corrections, you run your
file through RUNOFF, and  your  document  is  again  in  the
proper  format.  Once  your  document  is  ready,  you  can
reproduce it either

        1.  Into another file,
        2.  On your terminal, or
        3.  On a line-printer.


By using RUNOFF you can update your documentation  and  make
it look presentable without extensive retyping.
```

This is an example of how you input a list using the .LIST and .LIST ELEMENT commands. (.LS and .LE)

EXAMPLE 2

```
.LC.NF.LS
.LE;^PAGE NUMBERS ON EVERY PAGE EXCEPT THE FIRST,
.LE;^SPACING 1 BETWEEN LINES,
.LE;^FILL AND JUSTIFY,
.LE;^TAB STOPS 9,17,25,33,41,49,57,65,
.LE;^LEFT MARGIN 0,
.LE;^RIGHT MARGIN 60,
.LE;^PAGE SIZE = ^WIDTH 60 CHARACTERS, ^LENGTH 58 LINES,
.ELS
```

After using RUNOFF it looks like this

```
1.  Page numbers on every page except the first,
2.  Spacing 1 between lines,
3.  Fill and justify,
4.  Tab stops 9,17,25,33,41,49,57,65,
5.  Left margin 0,
6.  Right margin 60,
7.  Page size = Width 60 characters, Length 58 lines,
```

This .NOTE command does all the centering for you.

EXAMPLE 3

```
.LOWER CASE.FLAG CAPITALIZE
.NOTE
"THE ONLY COMMANDS WHICH CANNOT HAVE ANOTHER COMMAND
ON THE SAME LINE ARE .<TITLE, .<SUBTITLE, AND .<INDEX BECAUSE
ALL CHARACTERS ARE MADE PART OF THE TEXT.
.END NOTE
```

This is how it appears after RUNOFF executes the commands.

NOTE

The only commands which cannot have another command on the same line are .TITLE, .SUBTITLE, and .INDEX because all characters are made part of the text.

The following is an example of using the .CHAPTER command. The source file is made up of only this one command to illustrate its effect.

EXAMPLE 4

```
.MAKE CHAP

*I.CHAPTER RUNOFF                SOURCE FILE
$$
*EX$$

.R RUNOFF                        RUNNING THROUGH RUNOFF

*CHAP
CHAP    1 PAGE                   NO ERRORS
*↑C

.TY CHAP.MEM                     REQUESTING TYPE OUT ON THE TERMINAL.
```

```
CHAPTER I  }
           }   OUTPUT
RUNOFF     }
```

Example 5 shows you how .HEADER LEVELs work with a chapter oriented document.

EXAMPLE 5

```
.LOWER CASE.FLAG CAPITALIZE
.CHAPTER EXAMPLE 5
.HL 1 FIRST LEVEL OF SECTION 1
^THE COMMAND .<HEADER <LEVEL STARTS A SECTION AT THE LEVEL
SPECIFIED AND TAKES THE FOLLOWING TEXT AS THE HEADER.##^THE
N CAN BE IN THE RANGE FROM 1 TO 5.
.HL 1 SECOND LEVEL OF SECTION 1
^THE SECTIONS ARE INCREMENTED BY 1, AND THE NUMBER IS
OUTPUT IN THE FORM I.J.K.L.M.##^IF THIS IS A CHAPTER
ORIENTED DOCUMENT, THE I IS THE CHAPTER NUMBER;#OTHERWISE, THE
I IS THE NUMBER OF THE .<HL 1 LEVEL.
.HL 1 THIRD LEVEL OF SECTION 1
^THIS COMMAND ACTS AS
.SKIP 2;.NOFILL
        .<BREAK .<TEST <PAGE 9;.<BLANK 3
.SKIP 2;.FILL
FOLLOWED BY THE SECTION NUMBER.###<HEADER <LEVEL<S 1 AND 2
END WITH A .<BREAK.###<HEADER <LEVEL<S 3, 4, AND 5 END WITH
A SPACE DASH SPACE COMBINATION (_#-_#).
```

CHAPTER 1

EXAMPLE 5

1.1 FIRST LEVEL OF SECTION 1

The command .HEADER LEVEL starts a section at the level specified and takes the following text as the header. The n can be in the range from 1 to 5.

1.2 SECOND LEVEL OF SECTION 1

The sections are incremented by 1, and the number is output in the form i.j.k.l.m. If this is a chapter oriented document, the i is the chapter number; otherwise, the i is the number of the .HL 1 level.

1.3 THIRD LEVEL OF SECTION 1

This command acts as

```
.BREAK .TEST PAGE 9;.BLANK 3
```

followed by the section number. HEADER LEVELs 1 and 2 end with a .BREAK. HEADER LEVELs 3, 4, and 5 end with a space dash space combination (#-#).

7.0 SUMMARY OF MESSAGES

RUNOFF conforms to the DECsystem-10 standard for system diagnostic messages and error codes. The following conventions are used in describing these messages:

CONVENTION	MEANING
dev	a legal device name.
file structure name	a legal file structure name.
file.ext	a legal filename and extension.
adr	a user address.
n	a number.
abc	a disk unit or drive.
x	an alphabetic character.
switch	a switch.

The formats of the messages are

```
? RNFXXX text
% RNFXXX text
[ RNFXXX text
```

where

```
?      = fatal error message.
%      = warning or advisory message.
[      = comment line.
RNF    = RUNOFF mnemonic.
XXX    = 3 letter mnemonic for the message.
text   = explanation of the message.
```

If you input RUNOFF commands incorrectly into your source file, you can receive one or several of the messages in the following list:

?RNFCJL CAN'T JUSTIFY LINE
The string of input between spaces (and end of line) is greater than the separation between left and right margins and therefore does not fit in the output even before any attempt to expand spaces.

?RNFDVN DUPLICATE VARIABLE NAME: ".command"
This variable command is attempting to declare a variable which has already been declared. This declaration will be ignored.

?RNFEFD END FOOTNOTE DOESN'T FOLLOW FOOTNOTE: ".command"
This end footnote command appears at a place in the file which is not immediately following a footnote command and the corresponding footnote data.

?RNFELD END LITERAL DOESN'T FOLLOW LITERAL: ".command"
This end literal command appears at a place in the file which is not immediately following a literal command and the corresponding literal text. It probably reflects that the count on the literal command is incorrect.

?RNFEVL EXCEEDING VARIABLE LIST: ".command"
Only a maximum of 20 variables can be declared. This command is attempting to declare the 21st variable. This and all further variable declarations will be ignored, although the message will not be repeated.

?RNFFIF FOOTNOTE INSIDE FOOTNOTE: ".command"
This command is attempting to start a footnote definition, only it occurs within a footnote definition, which is illegal. It probably indicates that the previous footnote definition was never terminated.

?RNFIBO INPUT BUFFER OVERFLOW
A string of characters has been input which is so long between spaces (or the right margin is so large) that it has overflowed the internal line buffer storage area.

?RNFIFT ILLEGAL IN FOOTNOTE: ".command"
This command is illegal within a footnote.

%RNFIIF ^x IGNORED IN INPUT FILE
Control characters are not normally allowed in the input file. This character was input and is being ignored. If it should be input, then declare ".CONTROL CHARACTERS" in order to make it legal.

?RNFILC ILLEGAL COMMAND: ".command"
This command is illegal for some reason. Most reasons are that a key word was not recognized or that an argument was out of range.

%RNFJEC JUNK AT END OF COMMAND: ".command"
The command, after all its arguments, still has some other characters which are not blanks or comments.

[RNFKCF nK CORE - FOOTNOTE]
Core was expanded because of growth in the footnote storage area. If this repeats indefinitely it probably indicates that the footnote was improperly terminated.

[RNFKCI nK CORE - INDEX]
Core was expanded because of growth in the index storage area. In a well indexed document this message should be output occasionally as processing progresses.

?RNFLDE LITERAL DOESN'T END WITH .END LITERAL: ".command"
After a counted literal, the next line is not an end literal command. This probably indicates that the count is wrong.

?RNFNEC NOT ENOUGH CORE nK
Core has been expanded to the limit of what the monitor is allowed to assign to this job. The processing is aborted at this point.

?RNFNFS NO FILE SPECIFIED
The user has specified some switches or an output file, but has not specified an input file.

%RNFNIA NEGATIVE INDENT ATTEMPTED
The sum of the indent and the left margin is less than zero, meaning that the line should start to the left of the left edge of the paper. Either the left margin has been missed or the indent is wrong. The indent might be implicit in a paragraph or table request. This message is output only once until the next left margin or SD command.

%RNFNIC ANOTHER n NEGATIVE INDENTS COUNTED
This message indicates how many additional negative indents were discovered since the last NIA message.

?RNFNID NO INPUT DEVICE
The user has failed to specify either an input device or file.

?RNFODE OUTPUT ERROR xxxxxx
The output file has an I/O error whose octal code is included in the message.

%RNFSPO SUBPAGE OVERFLOW
While incrementing the subpage counter, it got larger than 26. This probably indicates that the end subpage command is missing.

%RNFTFE TOO FEW END COMMANDS
When the end of the input was reached there were not the same number of end (or end list or end note) commands as there had been list and note commands. This probably indicates that an end command is missing.

?RNFTMI INSUFFICIENT CORE FOR COMMAND
The user has specified so many input file specifications that RUNOFF could not fit them into core. The command should be split into several commands.

?RNFTMR TOO MANY RANGES
The user has specified too many IRANGE or ORANGE pairs to fit in the storage space assigned (20 pairs each). Fewer ranges should be specified.

?RNFTMV TOO MANY /VARIANTS
The user has specified too many VARIANTS in the command. Only 20 variants can be specified in one command.

%RNFTNN TOO MANY NESTED NOTES
More than 6 nested notes and lists has occurred. This probably indicates that one or more end commands is missing.

?RNFUKV UNKNOWN VARIABLE: ".command"
On an if, ifnot, else, or endif command a variable was referenced which was not declared in a variable command. This usually indicates a spelling error or a missing variable command.

%RNFUME UNMATCHED END COMMAND
More end commands have occurred than list or note commands.

?RNFYVZ /VARIANT VALUE ZERO
In a VARIANT switch, the value was null or zero. Variants always have names.

APPENDIX

SUMMARY OF RUNOFF INPUT FILE COMMANDS

COMMANDS	FUNCTION
.APPENDIX	Start next appendix with rest of line as name.
.AUTOPARAGRAPH	Treat leading spaces as new paragraph.
.AUTOTABLE	Treat lines without leading spaces as new paragraph.
.BEGIN BAR	Start a change bar.
.BLANK n	Skip n lines.
.BREAK	Start new output line.
.CENTER n or .CENTRE n	Center the next line around column n/2.
.CHAPTER	Start new chapter with the rest of line as name.
.COMMENT	Ignore this command.
.CONTROL CHARACTERS	Allow control characters.
.DISABLE BAR	Set to ignore change bars.
.DO INDEX	Output index with rest of line as title.
.ELSE	Change sense of IF/IFNOT.
.ENABLE BAR	Set to allow change bars.
.ENDIF name	End conditional input.
.END BAR	End change bar.
.END FOOTNOTE	Terminate a footnote definition.
.END LIST	End a list.
.END LITERAL	Terminate a literal block of text.

APPENDIX (Cont.)

COMMANDS	FUNCTION
.END NOTE	Terminate a NOTE command.
.END SELECTION	Stop selection until single line prefix.
.END SUBPAGE	Stop subpage numbering (resumes page).
.FIGURE n	Make space for n line figure.
.FIGURE DEFERRED n	Same except may be on next page.
.FILL	Resume filling and justifying each line.
.FIRST TITLE	Include title on first page.
.FLAGS ALL	Enable existing flag characters.
.FLAGS type ch	Change flag character to ch.
.FOOTNOTE n	Start n line footnote.
.HEADER x	Issue "page" in x (UPPER,LOWER,MIXED) case.
.HEADER LEVEL n	Start section at level n (1-5);rest is name.
.IF	Start conditional input if VARIANT name.
.IFNOT name	Start conditional input if not VARIANT name.
.INDENT n	Indent next line.
.INDEX	Insert rest of this line in index.
.JUSTIFY	Resume justifying text.
.LEFT n	Start next line n columns from left margin.
.LEFT MARGIN n	Set left margin.
.LIST n	Start list of items with spacing n.
.LIST ELEMENT	Start of item in a list.

APPENDIX (Cont.)

COMMANDS	FUNCTION
.LITERAL n	Start a literal block of text n lines long.
.LOWER CASE	Start footnotes and text in lower case.
.NO AUTOPARAGRAPH	Stop autoparagraph mode.
.NO AUTOTABLE	Stop autotable mode.
.NO CONTROL CHARACTERS	Do not allow control characters.
.NO FILL	Stop fill and justify.
.NO FLAGS ALL	Disable existing flag characters except . and !
.NO FLAGS type	Do not use flag characters type.
.NO HEADER	Suppress page headers.
.NO JUSTIFY	Stop justifying.
.NO NUMBER	Stop page numbering.
.NO PAGING	Stop splitting into pages.
.NO PERIOD	Stop double spacing after period, exclamation point, question, etc.
.NO SELECTION	Accept all text as input.
.NO SPACE	Suppress space on this end of line.
.NO SUBTITLE	Suppress subtitles.
.NOTE text	Start indented note with heading "text" centered.
.NUMBER n	Resume page numbering at page n.
.NUMBER APPENDIX n	Set chapter to appendix n.
.NUMBER CHAPTER n	Set chapter number to n.

APPENDIX (Cont.)

COMMANDS	FUNCTION
.NUMBER INDEX	Set chapter number to "INDEX".
.NUMBER LEVEL a, b, c, . . .	Set next HEADER LEVEL TO a, b, c, . . .
.NUMBER LIST d, c	Set list counter depth d to c.
.NUMBER PAGE n	Resume page numbering at page n.
.NUMBER SUBPAGE ch	Set subpage number to ch (A-Z).
.PAGE	Start new page.
.PAGE SIZE n, m .PAPER SIZE n, m	Paper is n lines by m columns. Paper is n lines by m columns.
.PAGING	Resume breaking into pages.
.PARAGRAPH n, v, t	Start new paragraph (.I n, .S v, .TP t).
.PERIOD	Double space after . ! ? : ;
.PRINT INDEX	Start printing index.
.RIGHT n	Right adjust next line n columns left of the margin.
.RIGHT MARGIN n	Set right margin.
.SELECTION string	Set selection string.
.SKIP n	Skip n*spacing lines.
.SPACING n	Set spacing (default=1).
.STANDARD n	Standard setup of width n.
.SUBINDEX	Index with " " used to delimit sub-indices.
.SUBPAGE	Start subpage numbering.
.SUBTITLE or .SUBTTL	Use rest of line as subtitle.

APPENDIX (Cont.)

COMMANDS	FUNCTION
.TAB STOPS n, n, . . .	Set tab stops.
.TEST PAGE n	Skip to new page if fewer than n lines left.
.TITLE	Use rest of line as title.
.TYPESET text	Send quoted text to TYPESET-10.
.UPPER CASE	Start footnotes and text in upper case.
.VARIABLE name ch ch	Declare variable with on/off flags ch, ch.

INDEX

MASTER INDEX