RT–11 System Utilities Manual
Part II

Order Number AA–PF6TA–TC

August 1991

This manual alphabetically describes 16 utilities beginning with MACRO–11 and ending with VTCOM (and including BATCH in an appendix).

Revision/Update Information: This manual supersedes the *RT–11 System Utilities Manual*, AA–M239C–TC.

Operating System: RT–11 Version 5.6

Digital Equipment Corporation
Maynard, Massachusetts

First Printing, March 1983
Revised, July 1984
Revised, November 1985
Reprinted, August 1989
Revised, August 1991

S1708

This document was prepared using VAX DOCUMENT, Version 1.2.

Contents

Tables

Preface

Audience

This manual is written for experienced users of the RT–11 operating system.

Document Structure

This manual alphabetically describes the following utilities:

MACRO–11

NITEST

ODT

PATCH

PIP

QUEUE

RESORC

RTMON

SIPP

SLP

SPLIT

SPOOL

SRCCOM

TRANSFER

VBGEXE

VTCOM

Appendix A describes the BATCH utility and Appendix B lists DCL command equivalents for all the CSI utility commands.

Conventions

The following conventions are used in this guide. Op

Convention	Meaning
Braces ({ })	In command syntax examples, braces enclose options that are mutually exclusive. You can choose only one option from the group of options that appear in braces.
Brackets ([ ])	Square brackets in a format line represent optional parameters, qualifiers, or values, unless otherwise specified.
lowercase characters	In command syntax examples, lowercase characters represent elements of a command for which you supply a value. For example: `.ASSIGN dev: WF` $\boxed{\text{RET}}$
UPPERCASE characters	In command syntax examples, uppercase characters represent elements of a command that should be entered exactly as given.
number. number$_{10}$	A number followed by a period or the subscript ten indicates a decimal number.
number number$_8$	A number without a decimal point (period) or followed by the subscript eight is an octal number, unless otherwise indicated. For example, 128. or 128_{10} is 128 (decimal) and 126 or 126_8 is 126 (octal).
$\boxed{\text{RET}}$	$\boxed{\text{RET}}$ in examples represents the RETURN key. Unless the manual indicates otherwise, terminate all commands or command strings by pressing $\boxed{\text{RET}}$.
$\boxed{\text{CTRL/}x}$	$\boxed{\text{CTRL/}x}$ indicates a control key sequence. While pressing the key labeled Ctrl, press another key. For example: $\boxed{\text{CTRL/C}}$.
Italics	*Italics* indicate a book title, information and error messages quoted in paragraphs, syntax elements of a command line when referenced in a paragraph, or, occasionally, a word highlighted in a paragraph because of its importance.

Associated Documents

Basic Books

- *Introduction to RT–11*
- *Guide to RT–11 Documentation*
- *RT–11 Commands Manual*
- *PDP–11 Keypad Editor User's Guide*
- *PDP–11 Keypad Editor Reference Card*
- *RT–11 Quick Reference Manual*
- *RT–11 Master Index*
- *RT–11 System Message Manual*
- *RT–11 System Release Notes*

Installation Specific Books

- *RT–11 Automatic Installation Guide*
- *RT–11 Installation Guide*
- *RT–11 System Generation Guide*

Programmer Oriented Books

- *RT–11 IND Control Files Manual*
- *RT–11 System Macro Library Manual*
- *RT–11 System Subroutine Library Manual*
- *RT–11 System Internals Manual*
- *RT–11 Device Handlers Manual*
- *RT–11 Volume and File Formats Manual*
- *DBG–11 Symbolic Debugger User's Guide*

Chapter 16
MACRO–11 Assembler Utility (MACRO)

This chapter describes how to assemble MACRO–11 programs on the RT–11 operating system.

Output from the MACRO–11 assembler includes any or all of the following:

- A binary-object file—the machine-readable logical equivalent of the MACRO–11 assembly language source code
- A listing of the source input file
- A cross-reference file listing
- A table-of-contents listing
- A symbol-table listing

To use the MACRO–11 assembler, you should understand how to:

- Initiate and terminate the MACRO–11 assembler.
- Format command strings to specify files MACRO–11 uses during assembly.
- Assign temporary work files to nondefault devices, if necessary.
- Use file-specification options to override file-control directives in the source program.
- Interpret error codes.

The following sections describe these topics. See the *Introduction to RT–11* for an introductory explanation of how to assemble a MACRO–11 file, and see the *PDP–11 MACRO–11 Language Reference Manual* for a detailed description of MACRO–11.

Calling and Terminating the MACRO–11 Assembler

To call the MACRO–11 assembler from the system device, respond to the system dot prompt by typing:

```
.R MACRO  RET
```

When the assembler responds with an asterisk (*), it is ready to accept command string input. (You can also call the assembler using the keyboard monitor MACRO command; see the *RT–11 Commands Manual* for a description of this command.)

To terminate the MACRO–11 assembler while it is waiting for input, press CTRL/C once. This returns you to system monitor control. To halt the assembly process at any time after you have completed the command string (thus beginning an assembly) press CTRL/C twice. This returns control to the system monitor, and a system monitor dot prompt appears on the terminal.

To restart the assembly process, issue the R MACRO command in response to the system monitor prompt.

Command-Line Syntax

The assembler expects a command string consisting of the following items, in sequence:

1. Output file specifications
2. An equal sign (=)
3. Input file specifications

Format this command string as follows (punctuation is required where shown):

dev:obj,dev:list,dev:cref/s:type=dev:source-1,...,dev:source-n/s:type

where:

dev	specifies any valid RT–11 device for output; any file-structured device for input.
obj	specifies the binary object file that the assembly process produces; the dev for this file should not be TT or LP.
list	specifies the assembly and symbol-listing file that the assembly process produces.
cref	specifies the CREF temporary cross-reference file that the assembly process produces. (However, if you omit dev:cref, you can still get a cross-reference listing.)
/s:type	specifies a set of options and arguments. (See the MACRO option descriptions for explanations of these options and arguments.)
source-1 through **source-n**	specify MACRO–11 source files or MACRO library files. (These files contain the MACRO language programs to be assembled. You can specify as many as six source files.)

The following command string calls for an assembly that uses one source file plus the system MACRO library to produce an object file BINF.OBJ and a listing. The listing goes directly to the line printer.

```
*DK:BINF.OBJ,LP:=DK:SRC.MAC RET
```

All output file specifications are optional. The system does not produce an output file unless the command string contains a specification for that file.

The system determines the file type of an output file specification by its position in the command string. Use commas in place of files you wish to omit. For example, to omit the object file, you must begin the command string with a comma. The following command produces a listing, including cross-reference tables, but not binary object files.

```
*,LP:/C=(source file specification) RET
```

You need not include a comma after the final output file specification in the command string.

Table 16–1 lists the default values for each file specification.

Table 16–1: Default File Specifications

File	Default Device	Default File Name	Default File Type
Object	DK	Must specify	OBJ
Listing	Same as for object file	Must specify	LST
Cref	DK	Must specify	TMP
First source	DK	Must specify	MAC
Additional source	Same as for preceding source file	Must specify	MAC
System MACRO library	System device (SY)	SYSMAC	SML
User MACRO	DK, if first file; otherwise, same as for preceding source file	Must specify	MLB

Assigning the Temporary Work File

Some assemblies need more symbol table space than available memory can) contain. When this occurs the system automatically creates a temporary work file called WRK.TMP to provide extended symbol table space.

The default device for WRK.TMP is DK. To cause the system to assign a different device, enter the following command:

```
.ASSIGN dev: WF  RET
```

The dev parameter is the physical name of a file-structured device. The system assigns WRK.TMP to this device.

If possible, assign WF to the VM device. This can speed up the assembly process and save you some disk space.

MACRO Options

At assembly time you may need to override certain MACRO directives appearing in the source programs. You may also need to direct MACRO–11 on the handling of certain files during assembly. You can satisfy these needs by including special options in the MACRO–11 command string in addition to the file specifications. Table 16–2 lists the options and describes the effect of each.

The general format of the MACRO–11 command string is repeated below for your convenience:

```
dev:obj,dev:list,dev:cref/s:type=dev:source-1,...,dev:source-n/s:type
```

Table 16–2: MACRO Options

Option	Function
/C:type	Controls the contents of the cross-reference listing.
/D:type	Disables the object-file function; overrides the source program directive .DSABL.
/E:type	Enables the object-file function; overrides the source program directive .ENABL/L:type listing control, overrides the source program directive .LIST.
/L:type	Controls the listing; overrides the source program directive .LIST.
/M	Indicates the input file is the MACRO library file.
/N:type	Controls the listing; overrides the source program directive .NLIST.

The /M option affects only the particular source file specification to which it is directly appended in the command string.

Other options are unaffected by their placement in the command string. The /L option, for example, affects the listing file, regardless of where you place it in the command string.

The following section describes how to use the MACRO options.

MACRO Option Descriptions

Listing-Control Options (/L:type and /N:type)

With two options, /L:type and /N:type, you can specify the content and format of assembler listings. With these options, you can also override at assembly time the arguments of .LIST and .NLIST directives in a source program.

Specifying the /N option with no argument causes the system to list only the symbol table, table of contents, and error messages. Specifying the /L option with no arguments causes the system to ignore .LIST and .NLIST directives that have no arguments.

Arguments for the /L and /N Listing-Control Options

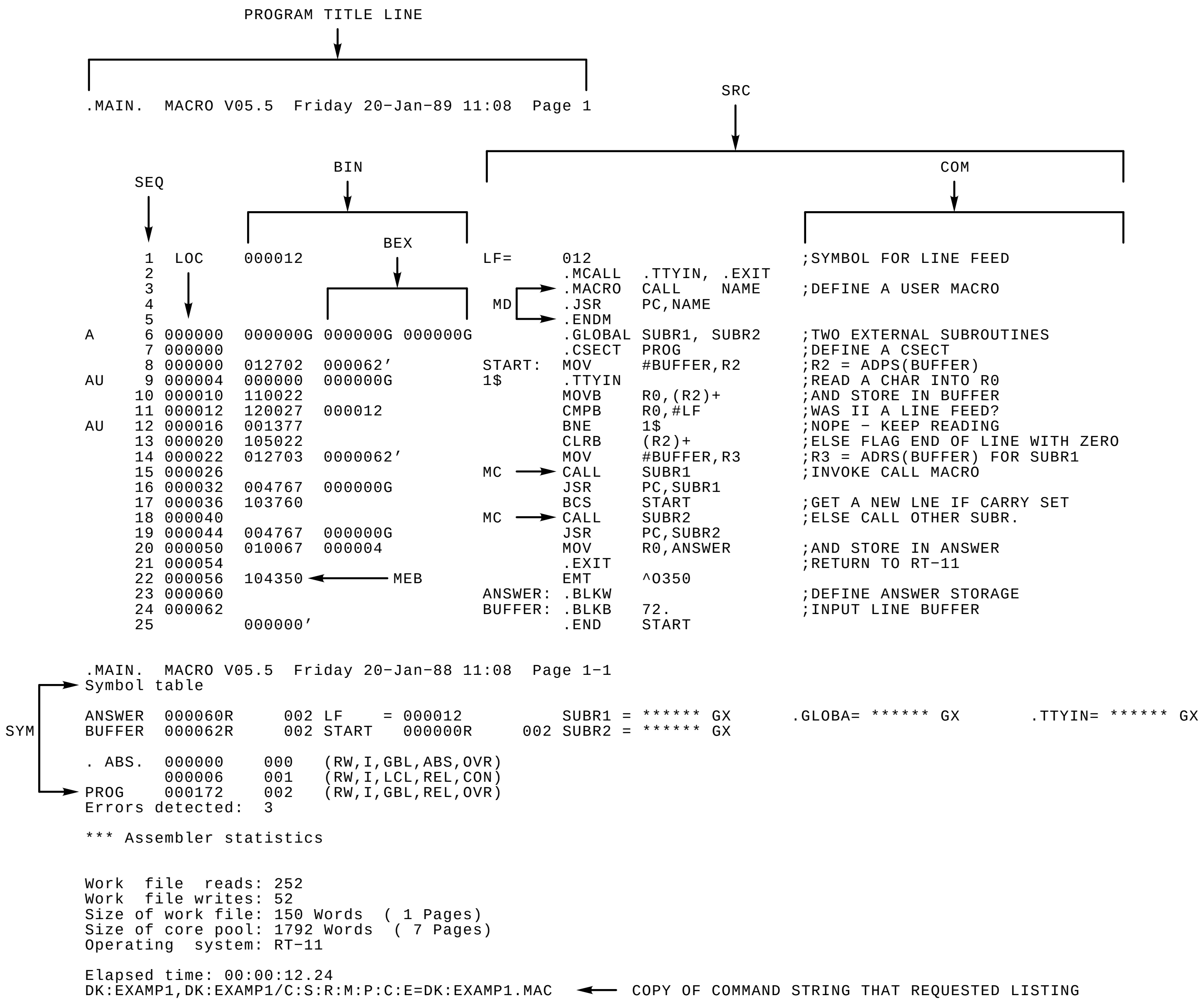
Argument	Default	Description
BEX	List	Binary extensions
BIN	List	Generated binary code
CND	List	Unsatisfied conditionals, .IF and .ENDC statements
COM	List	Comments
HEX	No List	Selects display radix of HEX
LD	No list	List control directives with no arguments
LOC	List	Address location counter
MC	List	Macro calls, repeat range expansion
MD	List	Macro definitions, repeat range expansion
ME	No list	Macro expansions
MEB	No list	Macro expansion binary code
SEQ	List	Source line sequence numbers
SRC	List	Source code
SYM	List	Symbol table
TOC	List	Table of contents
TTM	No list	132-column line printer format when not specified, terminal mode when specified

The following example command uses the /L and /N options. This command requests a listing of binary code throughout the assembly using the 132-column line-printer format, and suppresses the symbol-table listing.

```
*I,LP:/L:MEB/N:SYM=FILE RET
```

Figure 16–1 shows an assembly listing of a small program. In this listing, most of the /L and /N arguments in the preceding table label the sections of the listing which they control. For example, the SEQ argument controls the appearance of the source-line sequence numbers.

Figure 16–1: Sample Assembly Listing



```
.MAIN.  MACRO V05.5  Friday 20-Jan-89 11:08  Page 1


      1         000012                     LF=     012                      ;SYMBOL FOR LINE FEED
      2                                            .MCALL  .TTYIN, .EXIT
      3                                            .MACRO  CALL    NAME     ;DEFINE A USER MACRO
      4                                            .JSR    PC,NAME
      5                                            .ENDM
A     6 000000  000000G 000000G 000000G            .GLOBAL SUBR1, SUBR2     ;TWO EXTERNAL SUBROUTINES
      7 000000                                     .CSECT  PROG             ;DEFINE A CSECT
      8 000000  012702  000062'            START:  MOV     #BUFFER,R2       ;R2 = ADPS(BUFFER)
AU    9 000004  000000  000000G            1$      .TTYIN                   ;READ A CHAR INTO R0
     10 000010  110022                             MOVB    R0,(R2)+         ;AND STORE IN BUFFER
     11 000012  120027  000012                     CMPB    R0,#LF           ;WAS II A LINE FEED?
AU   12 000016  001377                             BNE     1$               ;NOPE - KEEP READING
     13 000020  105022                             CLRB    (R2)+            ;ELSE FLAG END OF LINE WITH ZERO
     14 000022  012703  0000062'                   MOV     #BUFFER,R3       ;R3 = ADRS(BUFFER) FOR SUBR1
     15 000026                                     CALL    SUBR1            ;INVOKE CALL MACRO
     16 000032  004767  000000G                    JSR     PC,SUBR1
     17 000036  103760                             BCS     START            ;GET A NEW LNE IF CARRY SET
     18 000040                                     CALL    SUBR2            ;ELSE CALL OTHER SUBR.
     19 000044  004767  000000G                    JSR     PC,SUBR2
     20 000050  010067  000004                     MOV     R0,ANSWER        ;AND STORE IN ANSWER
     21 000054                                     .EXIT                    ;RETURN TO RT-11
     22 000056  104350                             EMT     ^O350
     23 000060                             ANSWER: .BLKW                    ;DEFINE ANSWER STORAGE
     24 000062                             BUFFER: .BLKB   72.              ;INPUT LINE BUFFER
     25         000000'                            .END    START


.MAIN.  MACRO V05.5  Friday 20-Jan-88 11:08  Page 1-1
Symbol table

ANSWER  000060R     002 LF    = 000012          SUBR1 = ****** GX      .GLOBA= ****** GX       .TTYIN= ****** GX
BUFFER  000062R     002 START   000000R     002 SUBR2 = ****** GX

. ABS.  000000    000   (RW,I,GBL,ABS,OVR)
        000006    001   (RW,I,LCL,REL,CON)
PROG    000172    002   (RW,I,GBL,REL,OVR)
Errors detected:  3

*** Assembler statistics


Work  file  reads: 252
Work  file writes: 52
Size of work file: 150 Words  ( 1 Pages)
Size of core pool: 1792 Words  ( 7 Pages)
Operating  system: RT-11

Elapsed time: 00:00:12.24
DK:EXAMP1,DK:EXAMP1/C:S:R:M:P:C:E=DK:EXAMP1.MAC
```

Function-Control Options (/D:type and /E:type)

Two options, /E:type and /D:type, allow you to enable or disable functions at assembly time, and thus influence the form and content of the binary object file. These functions can override .ENABLE and .DSABL directives in the source program.

The following table summarizes the acceptable /E and /D function arguments, their normal default status, and the functions they control.

Arguments for /D and /E Function-Control Options

Argument	Default Mode	Function
ABS	Disable	Allows absolute binary output
AMA	Disable	Assembles all absolute addresses as relative addresses
CDR	Disable	Treats all source information beyond column 72 as commentary
CRF	Enable	Allows cross-reference listing; disabling this function inhibits CREF output if option /C is active
FPT	Disable	Truncates floating point values (instead of rounding)
GBL	Enable	Treats undefined symbols as globals
LC	Enable	Allows lowercase ASCII source input
LCM	Disable	Causes the MACRO–11 conditional assembly directives .if IDN and .if DIF to sense differences between uppercase and lowercase letters.
LSB	Disable	Allows local symbol block
MCL	Disable	Causes MACRO to search all MACRO libraries for a MACRO definition if an undefined op code is found
PNC	Enable	Allows binary output
REG	Enable	Allows mnemonic definitions of registers

For example, if you type the following commands the system assembles a file while treating columns 73 through 80 of each source line as commentary.

```
.R PIP  RET
*SRCPRG.MAC=CR:/A  RET
* CTRL/C
.R MACRO  RET
*,LP:=SRCPRG.MAC/E:CDR  RET
```

Because MACRO–11 is a two-pass assembler, you cannot read directly from any non-file-structured device. You must use PIP (or the keyboard monitor COPY command) to transfer input to a file-structured device before beginning the assembly.

Use either the function-control or listing-control option and arguments at assembly time to override corresponding listing or function-control directives in the source program. For example, assume that the source program contains the following sequence:

```
.NLIST MEB
  .
  .(MACRO references)
  .
.LIST MEB
```

In this example, you disable the listing of macro expansion binary code for some portion of the code and subsequently resume MEB listing. However, if you indicate /L:MEB in the assembly command string, the system ignores both the .NLIST MEB and the .LIST MEB directives. This enables MEB listing throughout the program.

Macro Library-File Designation Option (/M)

The /M option is meaningful only if appended to a source file specification. It designates its associated source file as a macro library.

If the command string does not include the standard system macro library SYSMAC.SML, the system automatically includes it as the first source file in the command string.

When the assembler encounters an .MCALL directive in the source code, it searches macro libraries according to their order of appearance in the command string. When it locates a macro record whose name matches that given in the .MCALL, it assembles the macro as indicated by that definition. Thus, if two or more macro libraries contain definitions of the same macro name, the macro library that appears rightmost in the command string takes precedence.

Consider the following command string:

```
* (output file specification)=ALIB.MLB/M,BLIB.MLB/M,XIZ
```

Assume that each of the two macro libraries, ALIB and BLIB, contain a macro called .BIG, but with different definitions. Then, if source file XIZ contains a macro call .MCALL .BIG, the system includes the definition of .BIG in the program as it appears in the macro library BLIB.

Moreover, if macro library ALIB contains a definition of a macro called .READ, that definition of .READ overrides the standard .READ macro definition in SYSMAC.SML.

Cross-Reference (CREF) Table-Generation Option (/C:type)

A cross-reference (CREF) table lists all or a subset of the symbols in a source program, identifying the statements that define and use the symbols.

Obtaining a Cross-Reference Table

To obtain a CREF table you must include the /C:type option in the command string. Usually you include the /C:type option with the assembly listing file specification, though you can place it anywhere in the command string.

The /C option can have six arguments, each separated from one another by a colon. These arguments are alphabetically summarized as follows.

Arguments for /C Option

Argument	Description
C	Lists control and program-section symbols
E	Lists error-code symbols and groupings
M	Lists MACRO symbolic names
P	Lists permanent symbols including instructions and directives
R	Lists register symbols
S	Lists user-defined symbols

NOTE

Specifying /C with no argument is equivalent to specifying /C:S:M:E. That special case excepted, you must explicitly request each CREF section by including its arguments.

You do not get a cross-reference file if you do not specify the /C option, even if you include a CREF file specification in the command string.

RT–11 places requested cross-reference tables after the MACRO assembly listing. Each table begins on a new page.

The following list describes the symbol contents of each of the six cross-reference tables. Each table begins on a page numbered to reference the table:

- **S-1**

  User-defined program symbols, beginning on page S-1.

  These are labels used in the program and symbols defined by a direct assignment statement.

- **R-1**

  Register equate symbols, beginning on page R-1.

  These normally include R0, R1, R2, R3, R4, R5, SP, and PC, unless the REG function has been disabled through a .DSABL REG directive or the /D:REG option. Also included are any other symbols that are defined in the program by the construct:

  ```
  symbol = %n
  ```

  where n represents the register number and has a value between 0 and 7.

- **M-1**

  MACRO symbols, beginning with page M-1.

  These are the symbols defined by .MACRO and .MCALL directives.

- **P-1**

  Permanent symbols, beginning with page P-1.

  These are all operation mnemonics and assembler directives.

- **C-1**

  Program sections, beginning with page C-1.

  These are the names you specify as operands of .CSECT or .PSECT directives. Also included are the default program sections produced by the assembler, the blank p-sect, and the absolute p-sect, .ABS.

- **E-1**

  Errors, beginning with page E-1.

  These are any errors, by error type, flagged by the assembler.

RT–11 displays symbols and also symbol values, control sections, and error codes, if applicable, beginning at the left margin of the page. References to each symbol are listed on the same line, left-to-right across the page. The system lists references in the form P-L, where P is the page in which the symbol, control section, or error code appears, and L is the line number on the page.

A number sign (#) next to a reference indicates a symbol definition. An asterisk (*) next to a reference indicates a destructive reference—that is, an operation that alters the contents of the addressed location.

The Sample Cross-Reference Table on the next page lists all the CREF tables the assembler produces for the MACRO–11 program displayed in Figure 16–1.

Sample Cross-Reference Tables

Note that these tables are combined onto one page. The page breaks after each table have been deleted to save space.

```
.MAIN. MACRO V05.5  Friday 25-Jan-90 11:08 Page S-1
Cross reference table (CREF V05.5)

.GLOBA   1-6
.TTYIN   1-9
ANSWER   1-20*     1-23#
BUFFER   1-8       1-14      1-24#
LF       1-1#      1-11
START    1-8#      1-17      1-25
SUBR1    1-6       1-15      1-16
SUBR2    1-6       1-18      1-19

.MAIN. MACRO V05.5  Friday 25-Jan-90 11:08 Page R-1
Cross reference table (CREF V05.5)

PC       1-15*     1-16*     1-18*     1-19*
R0       1-10      1-11      1-20
R2       1-8*      1-10*     1-13*
R3       1-14*

.MAIN. MACRO V05.5  Friday 25-Jan-90 11:08 Page M-1
Cross reference table (CREF V05.5)

.EXIT    1-2#      1-21
.TTYIN   1-2#
CALL     1-3#      1-15      1-18

.MAIN. MACRO V05.5  Friday 25-Jan-90 11:08 Page P-1
Cross reference table (CREF V05.5)

.BLKB    1-24
.BLKW    1-23
.CSECT   1-7
.END     1-25
.MACRO   1-3
.MCALL   1-2
BCS      1-17
BNE      1-12
CLRB     1-13
CMPB     1-11
EMT      1-21      1-22
JSR      1-15      1-16      1-18      1-19
MOV      1-8       1-14      1-20
MOVB     1-10

.MAIN. MACRO V05.5  Friday 25-Jan-90 11:08 Page C-1
Cross reference table (CREF V05.5)

         0-0
. ABS.   0-0
PROG     1-7

.MAIN. MACRO V05.5  Friday 25-Jan-90 11:08 Page E-1
Cross reference table (CREF V05.5)

A        1-6       1-9       1-12
U        1-9       1-12
```

Specifying a Cross-Reference Table File

You specify a cross-reference listing by means of the /C option. When you do so the system creates a temporary work file whose default name is DK:CREF.TMP.

You can explicitly specify the file to use as the temporary work file by naming it as the third output file. The system then uses your file specification instead of DK:CREF.TMP and deletes it automatically after producing the CREF listing. The following command string causes the system to use RK2:TEMP.TMP as the temporary CREF file:

```
*,LP:,RK2:TEMP.TMP=SOURCE/C  RET
```

Note that you must still include the /C option to control the form and content of the listing. Your specification for a cross-reference file is ignored if the /C option is not also present in the command string.

Another way to assign an alternate device for the CREF.TMP file is to enter the following command prior to entering R MACRO:

```
.ASSIGN dev: CF  RET
```

This method is preferred if you intend to do several assemblies, because it relieves you from having to include the dev:cref specification in each command string. If you enter the ASSIGN dev: CF command, and later include a CREF specification in a command string, the specification in the command string prevails for that assembly only.

If you assign CF to a physical device, that device also becomes the default device for the LINK temporary file CREF.TMP created when you use the LINK/GLOBAL (/N) option.

If your command string does not include a CREF file specification, the system automatically generates a temporary file on device DK. If you need to have a device other than DK contain the temporary CREF file, you must include the DEV:CREF field in the command string.

If the listing device is magtape, load the handler for that device before issuing the command string, using the monitor LOAD command (see the *RT–11 Commands Manual* for a description of that command).

MACRO–11 Error Codes

The MACRO–11 system prints diagnostic error codes as the first character of a source line on which the assembler detects an error. This error code identifies the type of error; for example, a code of M indicates a multiple definition of a label. Table 16–3 shows the error codes that might appear on an assembly listing. For detailed information on error code interpretation and debugging, see the *PDP–11 MACRO–11 Language Reference Manual*.

Table 16–3: MACRO–11 Error Codes

Error Code	Meaning
A	Addressing or relocation error. This code can be generated by any of the following: • A conditional branch instruction target that is too far above or below the current statement. Conditional branch targets must be within -128 to -127 (decimal) words of the instruction. • A statement that makes an invalid change to the current location counter. For example, a statement that forces the current location counter to cross a .PSECT boundary can generate this code. • A statement that contains an invalid address expression. For example, an absolute address expression that has a global symbol, relocatable value, or complex relocatable value can generate this code. The directives .BLKB, .BLKW, and .REPT must have an absolute value or an expression that reduces to an absolute value. • Separate expressions in the statement that are not separated by commas. • A global definition error. If .ENABL GBL is set, MACRO–11 scans the symbol table at the end of the first pass and marks any undefined symbols as global references. If one of these symbols is subsequently defined in the second pass, a general addressing error occurs. • A global assignment statement that contains a forward reference to another symbol. • An expression that defines the value of the current location counter and contains a forward reference. • An invalid argument for an assembler directive. • An unmatched delimiter or invalid argument construction.
B	Instruction or word data is being assembled at an odd address. The system increments the location counter by 1, and continues.
D	A nonlocal label is defined more than once, specifically in an earlier statement.
E	The .END assembler directive at the end of the source input is missing. The system supplies an .END statement and completes the current assembly pass.

Table 16–3 (Cont.): MACRO–11 Error Codes

Error Code	Meaning
I	MACRO–11 has detected one or more invalid characters. A question mark (?) replaces each invalid character on the assembly listing, and MACRO–11 continues after ignoring the character.
L	An input line is longer than 132 characters. In particular, this error occurs when the expansion of a macro causes excessive substitution of real arguments for dummy arguments.
M	A label is the same as an earlier label (multiple definition of a label). For example, two labels whose first six characters are identical can generate this error.
N	A number is not in the current program radix. MACRO–11 processes this number as a decimal value.
O	Op-code error. Exceeding the permitted nesting level for conditional assemblies causes this error. Attempting to expand a macro that remains unidentified after an .MCALL search can also generate this code.
P	Phase error. The definition or value of a label differs from one assembler pass to the next, or a local symbol occurs more than once in a local symbol block.
Q	Questionable syntax. For example, missing arguments, too many arguments, or an incomplete instruction scan can generate this error code.
R	Register-type error. For example, if the source program attempts an invalid reference to a register, the assembler can generate this error code.
T	Truncation error. A number that generates more than 16 bits in a word, or an expression in a .BYTE directive or trap instruction, can cause this error code.
U	Undefined symbol. The assembler assigns the undefined symbol a constant zero value.
Z	Incompatible instruction. This code is a warning that the instruction is not defined for all PDP–11 hardware configurations.

DCL Equivalents of MACRO Utility Operations

Table 16–4 lists the DCL MACRO command options that are equivalent to MACRO utility operations.

The first part of the table lists that part of the CSI MACRO command syntax that is equivalent to a DCL MACRO option. The rest of the table alphabetically lists all the MACRO options having DCL equivalents.

Table 16–4: DCL Equivalents of MACRO Utility Operations

CSI Command/Option	DCL Option
,filespec= (no 1st output filespec)	/NOOBJECT
filespec= (1st output filespec)	/OBJECT[:filespec]
,filespec= (2nd output filespec)	/LIST[:filespec]
filespec[size]= (1st output filespec)	/ALLOCATE:size
/C[:type[...:type]]	/CROSSREFERENCE[:type[...:type]]
/D:type[...:type]	/DISABLE:type[...:type]
/E:type[...:type]	/ENABLE:type[...:type]
/L:type	/SHOW:type
/M	/LIBRARY
/N:type	/NOSHOW:type

Note that the 1- and 3-letter *type* arguments to the utility options are the same arguments as those to the MACRO command qualifiers with three exceptions:

- The DCL /DISABLE and /ENABLE qualifier have a DBG type that the CSI command does not have.
- The CSI /D and /E options have a CRF and an MCL type that the DCL command does not have.

Chapter 17

Network Interconnect Test Utility (NITEST)

The Network Interconnect Test Utility (NITEST) is an unsupported utility that lets you verify that communications are possible between one machine on an Ethernet running RT–11 V5.2 and higher and another machine on the same Ethernet running NITEST or DECNET.

Building NITEST

NITEST is distributed in commented source form and is built by executing the following commands:

```
.MACRO NITEST RET
.LINK NITEST RET
```

Altering NITEST

Normally, NITEST sends the loopback request to the loopback assist multicast address. Changing the address specified, following label XBUFF, can let NITEST verify that communication is possible between two specific machines.

Running NITEST

Execute NITEST using one of the following commands:

```
.NITEST RET                (CCL, runs NITEST from device SY)

.R NITEST RET              (Runs NITEST from device SY)

.RUN NITEST RET            (Runs NITEST from device DK)

.RUN dev:NITEST RET        (Runs NITEST from device dev)
```

Using NITEST

When NITEST is started, it reports the physical address of the Ethernet interface board installed in the machine. It then indicates that loopback assist is enabled and prompts the user to press RETURN; for example:

```
.R NITEST  RET
   Station address = xx-xx-xx-xx-xx-xx
   Loopback assist is enabled
   Type <RETURN> to test:
```

If you press RETURN in response to the prompt, NITEST sends a loopback request message to the loopback assist multicast address. If NITEST has been altered as described above, NITEST sends a request to a station's physical address.

If NITEST receives no response to the loopback request message within 2 seconds, NITEST reports it and returns to the prompt; for example:

```
   Type <RETURN> to test:  RET
   ?NITEST-W-No Response
   Type <RETURN> to test:
```

If it receives a response, NITEST:

- Reports the address of the responding station.
- Compares the received data with the transmitted data and reports that the data is correct or corrupted.
- Displays the prompt.

For example:

```
   Type <RETURN> to test:  RET
   ?NITEST-I-Response received from xx-xx-xx-xx-xx-xx, data correct
   Type <RETURN> to test:
```

In the following example, NITEST reports that the data is corrupted:

```
   Type <RETURN> to test:  RET
   ?NITEST-I-Response received from xx-xx-xx-xx-xx-xx, data corrupt
   Type <RETURN> to test:
```

NITEST Operation

When executing, NITEST responds to loopback requests (protocol 90-00) sent to the station's physical address or to the loopback assist multicast address.

NITEST Error Messages

?NITEST-I-Response received from xx-xx-xx-xx-xx-xx, data correct

A response was received from station xx-xx-xx-xx-xx-xx, and the received data matched the transmitted data. Indicates a successful test.

?NITEST-I-Response received from xx-xx-xx-xx-xx-xx, data corrupt

A response was received from station xx-xx-xx-xx-xx-xx, but the received data did not match the transmitted data.

?NITEST-U-Enable multicast address error

An error occurred in enabling the loopback assist multicast address. See the ENABLE MULTICAST ADDRESS spfun described in the Ethernet handler documentation.

?NITEST-U-Enable protocol error

An error occurred in enabling the loopback protocol. See the errors for the ENABLE PROTOCOL spfun described in the Ethernet handler documentation.

?NITEST-U-Invalid device

The handler required for Ethernet access (NC for PRO-300 series processors, NQ for Q-bus processors, and NU for unibus processors) was not found in the monitor device tables. Install the required handler and run the test again.

?NITEST-U-Lookup error

An error occurred while attempting to open a channel to the Ethernet handler. See the errors for a .LOOKUP.

?NITEST-U-No queue element

A programmed request requiring a queue element was rejected because there was no queue element. This message should not occur.

?NITEST-U-Unit allocation error

An error occurred in allocating a unit of the Ethernet handler. See the errors for the ALLOCATE UNIT spfun described in the Ethernet handler documentation.

?NITEST-W-No Response

Following transmission of a loopback request, there was no response within 2 seconds. Hardware problems may be affecting transmission or reception; check the interface. There may be no stations on the Ethernet, or none which respond to loopback messages.

Chapter 18
On-Line Debugging Technique (ODT)

The On-Line Debugging Technique (ODT) Utility aids in debugging assembly language programs by:

- Displaying the contents of any location for examination or alteration.
- Running all or any portion of an object program using the breakpoint feature.
- Searching the object program for specific bit patterns.
- Searching the object program for words that refer to a specific word.
- Calculating offsets for relative addresses.
- Filling a single word, block of words, byte or block of bytes with a designated value.

Make sure you have an assembly listing and a link map available for the program you want to debug with ODT.

You can make minor on-line corrections to the program during the debugging session. Then you can verify the corrections by executing the program under the control of ODT. If you need to make major changes, such as adding a missing subroutine, note them on the assembly listing and incorporate them in a new assembly.

See the *RT–11 System Internals Manual* for information about debugging interrupt service routines, device handlers, multiterminal jobs, extended memory and virtual jobs. See also the *DBG–11 Symbolic Debugger User's Guide* for a description of the DBG–11 symbolic debugging package that lets you interactively debug an assembly-language program. This is a more powerful debugger than ODT.

Virtual On-Line Debugging Technique (VDT)

The Virtual On-Line Debugging Technique (VDT) utility is a variation of ODT. ODT accesses the hardware directly for terminal I/O; VDT uses RT–11 system services for terminal I/O. ODT cannot be used:

- On Professional 350/380 processors
- On a terminal line supported under a multi-terminal generated monitor
- With a virtual job

VDT can be used in the preceding three situations. To use VDT, follow all the instructions in this chapter for ODT except replace ODT with VDT in the LINK command.

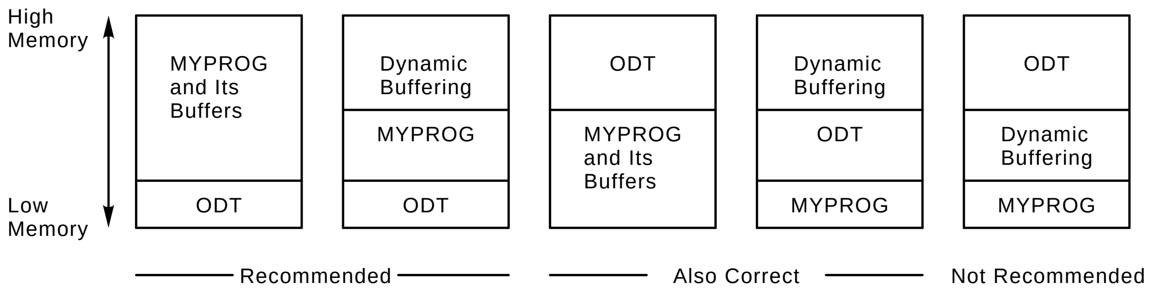
Calling and Using ODT

ODT is a relocatable object module. You can link ODT with your program (using the RT–11 Linker) for an absolute area in memory and load it with your program.

Link ODT low in memory relative to the program, but if you do link ODT high in memory, be sure that the buffer space for your program is contained within program bounds. Otherwise, if your program uses dynamic buffering, program execution may destroy ODT in memory. Figure 18–1 shows the relationship between ODT and the program MYPROG in memory.

To link ODT low in memory relative to your program, the program must declare a *named* PSECT by using the .PSECT directive. Because the linker orders *blank* PSECTs below named PSECTs in memory, your program should declare a named PSECT so that ODT will be linked lower in memory than your program.

Figure 18–1: Linking ODT with a Program



For example, if you include the directive .PSECT MYPROG in the program MYPROG, the following command will cause the linker to link ODT low in memory relative to MYPROG, and create the executable module MYPROG.SAV:

```
.LINK/DEBUG/MAP:TT: MYPROG RET
```

After it has been loaded in memory with your program, ODT has three legal start or restart addresses.

- The lowest (O.ODT) address, used for normal entry, retaining the current breakpoints.

  The system uses as an absolute address the address of the entry point O.ODT shown in the linker load map.

- The next (O.ODT+2) is a restart address that clears all breakpoints and reinitializes ODT, saving the general registers and clearing the relocation registers.
- The last address (O.ODT+4) is used to reenter ODT. A REENTER saves the processor status and general registers, and removes the breakpoint instructions

from your program. ODT displays the bad entry (BE) error message. Breakpoints that were set are reset by the next ;G command. (;P is invalid after a BE message.) The ;G and ;P commands for running the program are explained later in this chapter.

If you link ODT with an overlay-structured file, it should reside in the root segment so that it will always be in memory. Remove all breakpoints from the current overlay segment before execution proceeds to another overlay segment. A breakpoint inserted in an overlay is destroyed if it is overlaid during program execution.

Examples

1. This example command links ODT low in memory relative to MYPROG, creating the executable module MYPROG.SAV. Running MYPROG causes ODT to start automatically:

```
.LINK/MAP:TT:/DEBUG MYPROG
RT-11 LINK   V08.00       Load Map         Thursday  04-Nov-82 14:15  Page 1
MYPROG           .SAV     Title:  ODT      Ident:  05.00

Section  Addr    Size     Global  Value    Global  Value    Global  Value

. ABS.   000000  001000 = 256.    words   (RW,I,GBL,ABS,OVR)
$ODT$    001000  006152 = 1589.   words   (RW,I,LCL,REL,CON)
                         O.ODT    001232
PROG     007152  002052 = 533.    words   (RW,I,LCL,REL,CON)
                         START    007152

Transfer address = 001232, High limit = 011222 = 2377. words

.

.R MYPROG

 ODT  V05.00
*
```

2. This example command links MYPROG low in memory relative to ODT and specifies O.ODT as the transfer address. Running MYPROG causes ODT to start automatically. The advantage of using this method is that MYPROG is loaded at its normal, execution-time address:

```
.LINK/MAP:TT: MYPROG,ODT/TRANSFER
Transfer symbol? O.ODT
RT-11 LINK   V08.00       Load Map         Thursday  4-Nov-82 14:15  Page 1
MYPROG           .SAV     Title:  ODT      Ident:  05.00
Section  Addr    Size     Global  Value    Global  Value    Global  Value

. ABS.   000000  001000 = 256.    words   (RW,I,GBL,ABS,OVR)
PROG     001000  002052 = 533.    words   (RW,I,LCL,REL,CON)
                         START    001000
$ODT$    003052  006152 = 1589.   words   (RW,I,LCL,REL,CON)
                         O.ODT    003304

Transfer address = 003304, High limit = 011222 = 2377. words

.

.R MYPROG

 ODT  V05.00
*
```

3. This example command is similar to the previous example, except that execution does not automatically begin with ODT. When you start the program (MYPROG, in this case), you must specify the address of O.ODT as shown in the link map:

   ```
   .LINK/MAP:TT: MYPROG,ODT
   RT-11 LINK  V08.00      Load Map         Thursday    04-Nov-82 14:15
   Page 1
   MYPROG          .SAV    Title:  ODT      Ident:  05.00
   Section  Addr   Size    Global  Value    Global  Value    Global  Value

   . ABS.   000000 001000 = 256.   words   (RW,I,GBL,ABS,OVR)
   PROG     001000 002052 = 533.   words   (RW,I,LCL,REL,CON)
                           START   001000
   $ODT$    003052 006152 = 1589.  words   (RW,I,LCL,REL,CON)
                           O.ODT   003304

   Transfer address = 003304, High limit = 011222 = 2377. words

   .GET MYPROG

   .START 3304

    ODT  V05.00
   *
   ```

4. This example command links ODT with a bottom address of 4000, then loads ODT.SAV and MYPROG.SAV into memory. As in the previous example, when you start the program, you must specify the address of O.ODT as shown in the link map:

   ```
   .LINK/MAP:TT: ODT/BOTTOM:4000
   RT-11 LINK  V08.00      Load Map         Thursday    04-Nov-82 14:15
   Page 1
   ODT      .SAV   Title:  ODT     Ident:  05.00        /B:004000
   Section  Addr   Size    Global  Value   Global   Value  Global Value

   . ABS.   000000 004000 = 1024.  words  (RW,I,GBL,ABS,OVR)
   $ODT$    004000 006152 = 1589.  words  (RW,I,LCL,REL,CON)
                           O.ODT   004232

   Transfer address = 004232, High limit = 012150 = 2612. words

   .GET ODT.SAV

   .GET MYPROG.SAV

   .START 004232

    ODT  V05.00
   *
   ```

5. You can restart ODT by specifying O.ODT+2 as the start address. This reinitializes ODT and clears all breakpoints. For example:

   ```
   .START 4234
   *
   ```

6. You can reenter ODT by specifying O.ODT+4 as the start address. For example:

   ```
   .START 4236

   BE004242
   *
   ```

If ODT is waiting for a command, pressing CTRL/C calls the keyboard monitor. The monitor responds with a ^C on the terminal and waits for a command. (You can use the REENTER command to reenter ODT only if your program has set the reenter bit and ODT is linked high in memory relative to the program; otherwise, ODT is reentered at address O.ODT+6.)

If you press CTRL/U during a search display, the search terminates and ODT displays an asterisk prompt.

DCL Equivalents of ODT Operations

ODT is not accessible through DCL commands.

Relocation

When the assembler produces a relocatable object module, the base address of the module is assumed to be location 000000. The addresses of all program locations, as shown in the assembly listing, are relative to this base address. After you link the module, many of the values and all of the addresses in the program will be incremented by a constant whose value is the actual absolute base address of the module after it has been relocated. This constant is called the *relocation bias* for the module. Since a linked program may contain several relocated modules, each with its own relocation bias, and since, in the process of debugging, these biases will have to be continually subtracted from absolute addresses to relate relocated code to assembly listings, ODT provides automatic relocation.

The basis of automatic relocation is the eight relocation registers, numbered 0 through 7. You can set them to the values of the relocation biases at different times during debugging. Obtain relocation biases by consulting the link map. Once you have set a relocation register, ODT uses it to relate relative addresses to absolute addresses. For more information on the relocation process, see the Linker chapter.

ODT evaluates a relocatable expression as a 16-bit, six-digit (octal) number. You can type an expression in any one of the three forms presented in Table 18–1.

Table 18–1: Forms of Relocatable Expressions (r)

Form	Expression	Value of r
A	k	The value of k.
B	n,k	The value of k plus the contents of relocation register n. (If the n part of this expression is greater than 7, ODT uses only the last octal digit of n.)
C	C or C,k or n,C or C,C	Whenever you type the letter C, ODT replaces C with the contents of a special register called the constant register. (This value has the same role as the k or n that it replaces. The constant register is designated by the symbol $C and may be set to any value.)

Relocation register commands are discussed in detail later in the chapter. In Table 18–1, the symbol *n* stands for an integer in the range 0 through 7 inclusive, and the symbol *k* stands for an octal number up to six digits long, with a maximum value of 177777. If you type more than six digits, ODT takes the last six digits typed, truncated to the low-order 16 bits. If the symbol *k* is prefixed by a minus sign, its value is the two's complement of the number typed. For example:

k (number typed)	Values
1	000001
–1	177777
400	000400
–177730	000050
1234567	034567

Commands and Functions

ODT prompts for a command by displaying an asterisk on the terminal screen. You can issue most ODT commands in response to the asterisk:

- Examine a word and change it.
- Run the object program in its entirety or in segments.
- Search memory for specific words or references to them.

Display Formats

Normally, when ODT displays addresses it attempts to display them in relative form (Form B in Table 18–1). ODT looks for the relocation register whose value is closest to, but less than or equal to, the address to be displayed. It then displays the address relative to the contents of the relocation register. However, if no relocation register fits the requirement, the address is displayed in absolute form. Since the relocation registers are initialized to –1 (the highest number), the addresses initially display in absolute form. If you change the contents of any relocation register, it can then, depending on the command, qualify for relative form.

For example, suppose relocation registers 1 and 2 contain 1000 and 1004 respectively, and all other relocation registers contain much higher numbers. In this case, the following sequence might occur (the slash command causes the contents of the location to be displayed; pressing LINE accesses the next sequential location):

```
*1000;1R                        ;sets relocation register 1 to 1000
*1,4;2R                         ;sets relocation register 2 to 1004
*774/000000  LF                 ;opens location 774
000776 /000000  LF              ;opens location 776
1,000000 /000000  LF            ;absolute location 1000
1,000002 /000000  LF            ;absolute location 1002
2,000000 /000000                ;absolute location 1004
```

The display format is controlled by the format register, $F. Normally, this register contains 0; ODT displays relative addresses whenever possible. You can open $F and change its contents to a nonzero value, however. In that case, all addresses will be displayed in absolute form. See the discussion on accessing internal registers later in this chapter.

Opening, Changing, and Closing Locations

An open location is one whose contents ODT displays for examination, making those contents available for change. In a closed location, the contents are no longer available for change. Several commands are used for opening and closing locations.

Any command (except for the slash and backslash commands) that opens an additional location causes the currently open location to be closed. You can change the contents of an open location by typing the new contents followed by a single-character command that requires no argument; that is, LINE, ^, RET, <—, @, >, or <.

Slash (/)

One way to open a location is to type its address followed by a slash. For example:

```
*1000/012746
```

This command opens location 1000 for examination and makes it ready to be changed.

If you do not want to change the contents of an open location, press RETURN to close the location. ODT displays an asterisk prompt and waits for another command. However, to change the word, type the new contents before issuing the command to close the location. For example:

```
*1000/012746 012345  RET
*
```

This command inserts the new value, 012345, in location 1000 and closes the location. ODT displays an asterisk, prompting for another command.

Used alone, the slash reopens the last location opened. For example:

```
*1000/012345 2340  RET
*/002340
```

This command opens location 1000, changes its address to 002340, and then closes the location. ODT displays an asterisk prompt for another command. Typing the / character reopens the last location opened and verifies its value.

Opening an additional location automatically closes the currently open location before opening the new location.

If you specify an odd numbered address with a slash, ODT opens the location as a byte, and then behaves as if you had typed a backslash.

Backslash (\)

ODT operates on bytes, as well as on words. Typing the address of the byte followed by a backslash character opens the byte and displays the byte value at the specified address, interprets the value as ASCII code, and displays the corresponding character (if possible) on the screen. ODT displays a *?* when it cannot interpret the ASCII value as a character:

```
*1001\101 =A
```

Typing just a backslash reopens the last open byte. If a word was previously open, the backslash reopens its even byte:

```
*1002/000004 \004 =?
```

LINE(LF)

If you press LINE when a location is open, ODT closes the open location and opens the next sequential location:

```
*1000/002340  LF
001002 /012740
```

In this example, pressing LINE caused ODT to display the address of the next location and its contents and to wait for further instructions. After the preceding operation, location 1000 is closed and 1002 is open. You can modify the open location by typing the new contents.

If a byte location was open, pressing LINE opens the next byte location.

Circumflex or Up-Arrow

If you type the circumflex (or up-arrow) when a location is open, ODT closes the open location and reopens the previous location. To continue from the preceding example:

```
*001002/012740 ^
001000 /002340
```

This command closes location 1002 and opens location 1000. You may modify the open location by typing the new contents.

If the opened location was a byte, then the circumflex opens the previous byte.

Underline or Back-Arrow (_ or <–)

If you type the underline (or back-arrow) to an open word, ODT interprets the contents of the currently open word as an address indexed by the program counter (PC) and opens the addressed location:

```
*1006/000006 _
001016 /000405
```

Notice in this example that the open location, 1006, is indexed by the PC as if it were the operand of an instruction with addressing mode 67 (PC relative mode).

You can modify the opened location before you press LINE, circumflex, or underline. Also, the new contents of the location will be used for address calculations using the underline command. For example:

```
*100/000222 4  LF        ;modifies to 4 and open next location
000102 /000111 6^        ;modifies to 6 and open previous location
000100 /000004 200_      ;changes to 200 and open location indexed
000302 /123456           ;by PC
```

Open the Addressed Location (@)

You can use the at (@) symbol to optionally modify a location, close it, and then use its contents as the address of the location to open next. For example:

```
*1006/001044 @            ;opens location 1044 next
001044 /000500

*1006/001044 2100@        ;modifies to 2100 and opens location
002100 /000167            ;2100
```

Relative Branch Offset (>)

Use the right-angle bracket (>) to modify a location, close it, and then use its low-order byte as a relative branch offset to the next word to be opened. For example:

```
*1032/000407  301>        ;modifies to 301 and interprets as a
000636 /000010            ;relative branch
```

Note that 301 is a negative offset (–77). ODT doubles the offset before it adds it to the PC; therefore, 1034+(–176)=636.

Return to Previous Sequence (<)

Use the left-angle bracket (<) to modify a location, close it, and then open the next location of the previous sequence that was interrupted by an underline, @, or right-angle bracket command. Note that underline, @, or right-angle bracket causes a sequence change to the open word. If a sequence change has not occurred, the left-angle bracket opens the next location the same as pressing LINE. This command operates on both words and bytes:

```
*1032/000407  301>        ;> causes a sequence change
000636 /000010 <          ;returns to original sequence
001034 /001040 @          ;@ causes a sequence change
001040 /000405 \005 = <   ;< now operates on byte
001035 \002 =? <          ;< acts like  LF
001036 \004 =?
```

Accessing General Registers 0–7

Open the program's general registers 0–7 with a command in the following format:

```
$n/
```

The symbol, *n*, is an integer in the range 0–7 that identifies the desired register. When you open these registers, you can examine them or change their contents by typing in new data, as with any addressable location. For example:

```
*$0/000033 RET            ;examines register 0 then closes it
*

*$4/000474 464 RET        ;opens  register  4, changes its contents
*                          ;to 000464, then closes the register
```

The preceding example can be verified by typing a slash in response to ODT's asterisk:

```
*/000464
```

You can use LINE, circumflex, or @ command when a register is open.

Accessing Internal Registers

The program's status register contains the condition codes of the most recent operational results and the interrupt priority level of the object program. Open it by typing $S. For example:

```
*$S/000311
```

$S identifies the address of the status register. In response to $S in the preceding example, ODT displays a 16-bit word, of which only the low-order eight bits are meaningful. Bits 0–3 indicate whether a *carry, overflow, zero,* or *negative* (in that order) has resulted, and bits 5–7 indicate the interrupt priority level (in the range 0–7) of the object program. (Refer to the *PDP–11 Processor Handbook* for the Status Register format.)

You can also use the $ to open certain other internal locations listed in the following table.

Internal Registers

Register	Contents
$B	First word of the breakpoint table
$M	Mask location for specifying which bits are to be examined during a bit pattern search
$P	Defines the operating priority of ODT
$S	Condition codes (bits 0–3) and interrupt priority level (bits 5–7)
$C	Constant register
$R	Relocation register 0, the base of the Relocation Register table
$F	Format register

Radix–50 Mode (X)

Many PDP–11 system programs employ the Radix–50 mode of packing certain ASCII characters three-to-a-word. You can use Radix–50 mode by specifying the MACRO .RAD50 directive. ODT provides a method for examining and changing memory words packed in this way with the X command.

When you open a word and type the X command, ODT converts the contents of the opened word to its three-character Radix–50 equivalent and displays these characters on the terminal. You can then type one of the responses from the following table.

Radix–50 Terminators

Response	Effect
RETURN key RETURN	Closes the currently open location.
LINE FEED key LINE	Closes the currently open location and opens the next one in sequence.
Circumflex ^	Closes the currently open location and opens the previous one in sequence.
Any three characters whose octal code is 040 (space) or greater	Converts the three characters into packed Radix–50 format. Valid Radix–50 characters for this response are: . $ Space 0 through 9 A through Z

If you type any other characters, the resulting binary number is unspecified; that is, no error message displays and the result is unpredictable. You must type exactly three characters before ODT resumes its normal mode of operation. After you type the third character, the resulting binary number is available to be stored in the opened location. Do this by closing the location in any one of the ways listed in the preceding table. For example:

```
*1000/042431 X=KBI CBA  RET
*1000/011421 X=CBA
```

After ODT converts the three characters to binary, the binary number can be interpreted in one of many different ways, depending on the command that follows. For example:

```
*1234/063337 X=PRO XIT/013704
```

Since the Radix–50 equivalent of XIT is 113574, the final slash in the example will cause ODT to open location 113574 if it is a valid address.

Breakpoints

The breakpoint feature helps you monitor the progress of program execution. You can set a breakpoint at any instruction that is not referenced by the program for data. When a breakpoint is set, ODT replaces the contents of the breakpoint location with a BPT trap instruction so that program execution is suspended when a breakpoint is encountered. Then the original contents of the breakpoint location are restored, and ODT regains control.

With ODT you can set up to eight breakpoints, numbered 0 through 7, at any one time. Set a breakpoint by typing the address of the desired location of the breakpoint followed by ;B. Thus, r;B sets the next available breakpoint at location *r*. (If all eight breakpoints have been set, ODT ignores the r;B command.) You may set or change specific breakpoints by the r;nB command, where *n* is the number of the breakpoint. For example:

```
*1020;B     ;sets breakpoint 0
*1030;B     ;sets breakpoint 1
*1040;B     ;sets breakpoint 2
*1032;1B    ;resets breakpoint 1
*
```

The ;B command removes all breakpoints. Use the ;nB command to remove only one of the breakpoints, where *n* is the number that identifies the breakpoint. For example:

```
*;2B       ;removes breakpoint 2
*
```

ODT keeps a table of breakpoints that you can access. The $B/ command opens the location containing the address of breakpoint 0. The next seven locations contain the addresses of the other breakpoints in order. You can sequentially open them by pressing the LINE key. For example:

```
*$B/001020  LF
001136 /001032  LF
001140 /007070  LF
001142 /007070  LF
001144 /007070  LF
001146 /001046  LF
001150 /001066  LF
001152 /007070
```

In this example, breakpoint 0 is set to 1020, breakpoint 1 is set to 1032, breakpoint 5 is set to 1046, and breakpoint 6 is set to 1066. The other breakpoints are not set.

Note that a repeat count in a proceed command (;P) refers only to the breakpoint that ODT most recently encountered. Execution of other breakpoints is determined by their own repeat counts.

Running the Program (r;G and r;P)

ODT controls program execution. There are two commands for running the program: r;G and r;P. The r;G command starts execution (go) and r;P continues (proceed) execution after halting at a breakpoint. For example:

```
*1000;G
```

This command starts execution at location 1000. The program runs until it encounters a breakpoint or until it completes. If it gets caught in an infinite loop, either restart or reenter it as explained in the section, Calling and Using ODT.

Upon execution of either the r;G or r;P command, the general registers 0–6 are set to the values in the locations specified as $0–$6. The processor status register is set to the value in the location specified as $S.

When ODT encounters a breakpoint, execution stops and ODT displays Bn; (where *n* is the breakpoint number), followed by the address of the breakpoint. You can then examine locations for expected data. For example:

```
*1010;3B            ;sets breakpoint 3 at location 1010
*1000;G             ;starts execution at location 1000
B3;001010           ;stops execution at location 1010
*
```

To continue program execution from the breakpoint, type ;P in response to ODT's last prompt (*).

When you set a breakpoint in a loop, you can allow the program to execute a specified number of times through the loop before ODT recognizes the breakpoint. Set a proceed count by using the r;P command. This command specifies the number of times the breakpoint is to be encountered before ODT suspends program execution (on the *kth* encounter). The count *k* refers only to the numbered breakpoint that most recently occurred. You can specify a different proceed count for the breakpoint when it is encountered:

```
B3;001010           ;halts execution at breakpoint 3
*1026;3B            ;resets breakpoint 3 at location 1026
*4;P                ;sets proceed count to 4 and
B3;001026           ;continues execution; the program loops
*                   ;through the breakpoint three times and halts on
                    ;the fourth occurrence of the breakpoint
```

Following the table of breakpoints is a table of proceed command repeat counts for each breakpoint. You can inspect these repeat counts by typing $B/ and nine line feeds. The repeat count for breakpoint 0 displays (the first seven line feeds cause the table of breakpoints to be displayed; the eighth types the single-instruction mode, explained in the next section, and the ninth line feed begins the table of proceed command repeat counts). The repeat counts for breakpoints 1 through 7 and the repeat count for the single-instruction trap follow in sequence. ODT initializes a proceed count to 0 before you assign it a value. After the command has been executed, it is set to –1. Opening any one of these provides an alternative way of changing the count. Once the location is open, you can modify its contents by typing the new contents and then pressing RETURN. For example:

```
.
.
.
nnnnnn /001036 LF          ;address of breakpoint 7
nnnnnn /006630 LF          ;single instruction address
nnnnnn /000000 15 LF       ;count for breakpoint 0; changes to 15
nnnnnn /000000 LF          ;count for breakpoint 1
.
.
.
nnnnnn /000000 LF          ;count for breakpoint 7
nnnnnn /nnnnnn            ;repeat count for single instruction
                          ;mode.
```

Both the address indicated as the single-instruction address and the repeat count for single-instruction mode are explained in the following section.

Single-Instruction Mode

With this mode, you specify the number of instructions to be executed before ODT suspends the program run. The proceed command, instead of specifying a repeat count for a breakpoint encounter, specifies the number of succeeding instructions to be executed. Note that breakpoints are disabled in single-instruction mode. The following table lists the single-instruction mode commands.

Single-Instruction Mode Commands

Command	Function
;nS	Enables single-instruction mode (*n* can be any digit and serves only to distinguish this form from the form ;S, which disables single-instruction mode). Breakpoints are disabled.
n;P	Proceeds with program run for next *n* instructions before reentering ODT. (If *n* is missing, it is assumed to be 1.) Trapping instructions and associated handlers can affect the proceed repeat count (see Programming Considerations, functional organization discussion.)
;S	Disables single-instruction mode.

When the repeat count for single-instruction mode is exhausted and the program suspends execution, ODT displays:

```
B8;nnnnnn
```

where *nnnnnnn* is the address of the next instruction to be executed. The $B breakpoint table contains this address following that of breakpoint 7. However, unlike the table entries for breakpoints 0–7, direct modification has no effect.

Similarly, following the repeat count for breakpoint 7 is the repeat count for single-instruction mode. You can modify this table entry directly. This is an alternative way of setting the single-instruction mode repeat count. In such a case, ;P implies the argument set in the $B repeat count table rather than an assumed 1.

Searches

With ODT you can search any specific portion of memory for bit patterns or references to a particular location.

Word Search (r;W)

Before initiating a word search, you must specify the mask and search limits. The location indicated by $M specifies the mask of the search. $M/ opens the mask register. The next two sequential locations (opened by LFs initially contain the lower and upper limits of the search. ODT examines in the search all bits set to 1 in the mask and ignores other bits.

You must then give the search object and the initiating command, using the r;W command, where *r* is the search object. When ODT finds a match (that is, each bit set to 1 in the search object is set to 1 in the word ODT searches over the mask range), the matching word displays. For example:

```
*$M/000000 177400 LF                ;tests high-order eight bits
r,nnnnnn--/000000  1000 LF          ;sets low address limit
r,nnnnnn--/000000  1040 RET         ;sets high address limit
*400;W                              ;initiates word search
001010 /000770
001034 /000404
*
```

In the preceding example, nnnnnn is an address internal to ODT; this location varies and is meaningful only for reference purposes. In the first line, the slash was used to open $M, which now contains 177400; the LINEs open the next two sequential locations, which now contain the upper and lower limits of the search.

In the search process, ODT performs an exclusive OR (XOR) with the word currently being examined and the search object; the result is ANDed to the mask. If this result is 0, a match has been found and ODT reports it on the terminal. Note that if the mask is 0, all locations within the limits display. This provides a convenient method for dumping all memory locations within given limits using ODT.

Pressing CTRL/U during a search display terminates the search.

Effective Address Search (r;E)

ODT provides a search for words that reference a specific location. Open the mask register only to gain access to the low- and high-limit registers. After specifying the search limits (as explained for the word search), type the command r;E (where *r* is the effective address) to initiate the search.

Words that are an absolute address (argument *r* itself), a relative address offset, or a relative branch to the effective address are displayed after their addresses. For example:

```
*$M/177400 LF                     ;opens mask register only to gain
r,nnnnnn /001000 1010 LF          ;access to search limits
r,nnnnnn /001040 1060  RET
*1034;E                          ;initiates search
001016 /001006                   ;relative branch
001054 /002767                   ;relative branch
*1020;E                          ;initiates a new search
001022 /177774                   ;relative address offset
001030 /001020                   ;absolute address
```

Pay particular attention to the reported effective address references. A word can have the specified bit pattern of an effective address without actually being used as one. ODT reports all possible references whether they are actually used or not.

Pressing CTRL/U during a search display terminates the search.

Constant Register (r;C)

You may want to convert a relocatable address into its value after relocation or convert a number into its two's complement and then to store the converted value into one or more places in a program. Use the constant register to perform this and other useful functions.

Typing r;C evaluates the relocatable expression to its six-digit octal value, displays the value on the terminal, and stores it in the constant register. Invoke the contents of the constant register in subsequent relocatable expressions by typing the letter C. Examples follow:

```
*-4432;C=173346         ;places the two's complement of 4432 in the
                        ;constant register

*6632/062701 C RET      ;stores the contents of the constant
                        ;register in location 6632

*1000;1R                ;sets relocation register 1 to 1000

*1,4272;C=005272        ;displays relative location 4272 as an
                        ;absolute location and stores it in the
                        ;constant register
```

Memory-Block Initialization (;F and ;I)

Use the constant register with the commands ;F and ;I to set a block of memory to a specific value. While the most common value required is 0, other possibilities are +1, –1, ASCII space.

When you type the command ;F, ODT stores the contents of the constant register in successive memory words, starting at the memory word address you specify in the lower search limit and ending with the address you specify in the upper search limit.

Typing the command ;I stores the low-order eight bits in the constant register in successive bytes of memory, starting at the byte address you specify in the lower search limit and ending with the byte address you specify in the upper search limit.

For example, assume relocation register 1 contains 7000, 2 contains 10000, and 3 contains 15000. The following sequence sets word locations 7000–7776 to 0, and byte locations 10000–14777 to ASCII spaces:

```
                                ;opens the mask register to gain
*$M/000000 LF                   ;access to search limits
r,nnnnnn /000000 1,0 LF         ;sets the lower limit to 7000
r,nnnnnn /000000 2,-2 RET       ;sets the upper limit to 7776
*0;C=000000                     ;sets the constant register to zero
*;F                             ;sets locations 7000-7776 to zero
```

```
*$M/000000 LF
r,nnnnnn/007000 2,0 LF         ;sets the lower limit to 10000
r,nnnnnn/007776 3,-1 RET       ;sets the upper limit to 14777
*40;C=000040                   ;sets the constant register to 40
                               ;(space)
*;I                            ;sets the byte locations
                               ;10000-14777
*                              ;to the value in the low-order
                               ;eight bits of the constant
                               ;register
```

Calculating Offsets (r;O)

Relative addressing and branching involve the use of an offset. An offset is the number of words or bytes forward or backward from the current location to the effective address. During the debugging session it is sometimes necessary to change a relative address or branch reference by replacing one instruction offset with another. ODT calculates the offsets in response to the r;O command.

The command r;O causes ODT to display the 16-bit and 8-bit offsets from the currently open location to address *r*. For example:

```
*346/000034 414;O 000044 022 22  RET
*/000022
```

This command opens location 346, calculates and displays the offsets from location 346 to location 414, changes the contents of location 346 to 22 (the 8-bit offset), and verifies the contents of location 346.

The 8-bit offset displays only if it is in the range -128(decimal) to 127(decimal) and the 16-bit offset is even, as shown in the previous example. In the next example, the offset of a relative branch is calculated and modified so that it branches to itself:

```
*1034/103421 1034;O 177776 377 \021 =?  377 RET
*/103777
```

Note that the modified low-order byte 377 must be combined with the unmodified high-order byte.

Relocation Register Commands

The use of the relocation registers is described briefly at the beginning of this chapter. At the beginning of a debugging session it is desirable to preset the registers to the relocation biases of those relocatable modules that will be receiving the most attention. Do this by typing the relocation bias, followed by a semicolon and the specification of relocation registers, using the following syntax:

r;nR

The symbol *r* may be any relocatable expression, and *n* is an integer in the range 0–7. If you omit *n*, it is assumed to be 0. For example:

```
*1000;5R        ;puts 1000 into relocation register 5
*5,100;5R       ;adds 100 to the contents
*               ;of relocation register 5
```

Once a relocation register is defined, you can use it to reference relocatable values. For example:

```
*2000;1R         ;puts 2000 into relocation register 1
*1,2176/002466   ;examines the contents of location 4176
*1,3712;0B       ;sets a breakpoint at location 5712
```

Sometimes programs may be relocated to an address below the one at which they were assembled; for instance, with PIC code (position-independent code), which is moved without using the linker. In this case, the appropriate relocation bias would be the two's complement of the actual downward displacement. One method for easily evaluating the bias and putting it in the relocation register is illustrated in the following example.

Assume a program assembled at location 5000 was moved to location 1000. Then the following sequence enters the two's complement of 4000 in relocation register 1:

```
*1000;1R
*1,-5000;1R
*
```

Relocation registers are initialized to -1 so that unwanted relocation registers never enter into the selection process when ODT searches for the most appropriate register.

To set a relocation register to -1, type ;nR. To set all relocation registers to -1, type ;R.

ODT maintains a table of relocation registers, beginning at the address specified by $R. Opening $R ($R/) opens relocation register 0. Successively pressing LINE opens the other relocation registers in sequence. When a relocation register is opened in this way, you can modify it as you would any other memory location.

The Relocation Calculators, n! and nR

When a location has been opened, it is often desirable to relate the relocated address and the contents of the location back to their relocatable values. To calculate the relocatable address of the opened location relative to a particular relocation bias, use the following syntax:

n!

The symbol *n* specifies the relocation register. This calculator works with opened bytes and words. If you omit *n*, the relocation register whose contents are closest to, but less than or equal to, the opened location is selected automatically by ODT. In the following example, assume that these conditions are fulfilled by relocation register 3, which contains 2000. Use the following command to find the most likely module that a given opened byte is in:

```
*2500\011 = !=3,000500
```

To calculate the difference between the contents of the opened location and a relocation register, use the following syntax:

nR

The symbol *n* indicates the relocation register. If you omit *n*, ODT selects the relocation register whose contents are closest to, but less than or equal to, the contents of the opened location. For example, assume the relocation bias stored in relocation register 1 is 7000:

```
*1,500/011032 1R=1,002032
```

The value 2032 is the content of 1,500, relative to the base 7000. The next example shows the use of both relocation calculators.

If relocation register 1 contains 1000, and relocation register 2 contains 2000, use the following command to calculate the relocatable addresses of location 3000 and its contents, relative to 1000 and 2000:

```
*3000/006410 1!=1,002000 2!=2,001000 1R=1,005410 2R=2,04410
```

ODT Priority Level ($P)

$P identifies a location in ODT that contains the interrupt (or processor) priority level at which ODT operates. If $P contains the value 377, ODT operates at the priority level of the processor at the time ODT is entered. Otherwise $P may contain a value between 0 and 7 corresponding to the fixed priority at which ODT operates.

To set ODT to the desired priority level, open $P. ODT displays the present contents, which you can then change:

```
*$P/000006 4 RET  ;lowers the priority to allow interrupts
*                  ;from the terminal
```

If you do not change $P, its value is seven.

You must set ODT's priority to 0 if you are using ODT in a foreground/background environment while another job is running.

ODT may not always service breakpoints that are set in routines that run at different priority levels. For example, a program running at a low priority can use a device service routine that operates at a higher priority level. If you set $P low, ODT waits for terminal input at a low priority. If an interrupt occurs from a high-priority routine, the breakpoints in the high-priority routine will not be recognized because they were removed when the earlier breakpoint occurred. Thus, interrupts that are set at a priority higher than the one at which ODT is running will be serviced, but any breakpoints will not be recognized. To avoid this problem, set breakpoints at one priority level at a time. That is, set breakpoints within an interrupt service routine, but not at mainline code level. For a more complete discussion of how the PDP–11 handles priority and interrupts, refer to the processor handbook for your particular machine. ODT disables all breakpoints in the program whenever it gains control. Breakpoints are enabled when ;P and ;G commands are executed. For example:

```
*$P/00007 5
*1000;B
*2000;B
*1000;G
B0;001000
*             ;an interrupt occurs and is serviced
```

If a higher-level interrupt occurs while ODT is waiting for input, the interrupt is serviced, and no breakpoints are recognized.

ASCII Input and Output (r;nA)

Inspect and change ASCII text by using a command of this syntax:

r;nA

The symbol *r* signifies a relocatable expression, and *n* is a character count. If you omit *n*, it is assumed to be 1. ODT displays *n* characters starting at location *r* followed by a RET LF combination. The following table lists responses and their effect.

ASCII Terminators

Response	Effect
RETURN	ODT displays a RET/LF combination followed by an asterisk prompt for another command.
LINE	ODT opens the byte following the last byte displayed.
Up to *n* characters of text	ODT inserts the text into memory, starting at location *r*. If you type exactly *n* characters, ODT responds with RET LF address *. If you type fewer than *n* characters, terminate that string with CTRL/U. ODT responds with ?^U LF RET LF address RET LF

Programming Considerations

Information in this section is not necessary for normal use of ODT. However, it will help you understand how ODT functions, even under difficult debugging circumstances.

Using ODT with Foreground/Background Jobs

It is possible to use ODT to debug programs written as either background or foreground jobs. In the background or under the SB monitor, you can link ODT with the program as illustrated in the first example in the chapter. To debug a program in the foreground area, Digital recommends running ODT in the background while the program to be debugged is in the foreground. Issue the following sequence of commands:

```
.FRUN PROG/P         ;loads the foreground program
LOADED AT nnnnnn     ;the first address of the job displays
.RUN ODT             ;runs ODT in the background
ODT V01.01           ;and sets a relocation register
*nnnnnn;0R           ;to the start of the job

*$F/000000 0         ;clears the format register to enable
*0,nnnnnn;0B         ;proper address displays
                     ;sets a breakpoint

*0;G                 ;starts the keyboard monitor again

.RESUME              ;starts the foreground job
```

Link the copy of ODT low enough so that it fits in memory with the foreground job.

NOTE

Because ODT uses its own terminal handler, it cannot be used with the display hardware. If GT ON is in effect, ODT ignores it and directs its input and output only to the terminal.

If you use ODT in a foreground/background environment while another job is running, set ODT's priority bit to 0 as follows:

```
*$P/000007 0 RET
```

This command puts ODT into the wait state at level 0, not at level 7. If you leave ODT's priority at 7, all interrupts (including clock) are locked out while ODT is waiting for terminal input.

Functional Organization

The internal organization of ODT is almost totally modularized into independent subroutines. The internal structure consists of three major functions: *command decoding, command execution,* and *utility routines*.

The command decoder interprets the individual commands, checks for command errors, saves input parameters for use in command execution, and sends control to the appropriate command execution routine.

The command execution routines take parameters saved by the command decoder and use the utility routines to execute the specified command. Command execution routines either return to the command decoder or transfer control to your program.

The utility routines are common routines such as SAVE–RESTORE and I/O. They are used by both the command decoder and the command executers.

Breakpoints

A breakpoint gives control to ODT whenever a program tries to execute the instruction at the selected address.

When a breakpoint is executed, ODT removes all the breakpoint instructions from the code so that you can examine and alter the locations. ODT then displays a message on the terminal in the form Bn;r,

where:

r	is the breakpoint address.
n	is the breakpoint number. ODT restores the breakpoints when execution resumes.

A major restriction in the use of breakpoints is that the program must not reference the word where a breakpoint was set since ODT altered the word. Avoid setting a breakpoint at the location of any instruction that clears the T-bit. For example:

```
MOV #240,177776        ;SET PRIORITY TO LEVEL 5
```

NOTE

Instructions that cause traps or returns from them (for example, EMT, RTI) are likely to clear the T-bit, because a new word from the trap vector or the stack is loaded into the status register.

A breakpoint occurs when a trace trap instruction (placed in your program by ODT) is executed. When a breakpoint occurs, ODT operates according to the following algorithm:

1. Sets processor priority to seven (automatically set by trap instruction).
2. Saves registers and sets up stack.
3. If internal T-bit trap flag is set, goes to step 13.
4. Removes breakpoints.
5. Resets processor priority to ODT's priority or user's priority.
6. Makes sure a breakpoint or single-instruction mode caused the interrupt.

7. If the breakpoint did not cause the interrupt, goes to step 15.
8. Decrements repeat count.
9. Goes to step 18 if nonzero; otherwise resets count to one.
10. Saves terminal status.
11. Types message about the breakpoint or single-instruction mode interrupt.
12. Goes to command decoder.
13. Clears T-bit in stack and internal T-bit flag.
14. Jumps to the go processor.
15. Saves terminal status.
16. Types BE (bad entry), followed by the address.
17. Clears the T-bit, if set, in the user status and proceeds to the command decoder.
18. Goes to the proceed processor, bypassing the TT restore routine.

Note that steps 1–5 take approximately 100 microseconds. Interrupts are not permitted during this time, because ODT is running at priority level 7.

ODT processes a proceed (;P) command according to the following algorithm:

1. Checks the proceed for validity.
2. Sets the processor priority to seven.
3. Sets the T-bit flags (internal and user status).
4. Restores the user registers, status, and program counter.
5. Returns control to the user.
6. When the T-bit trap occurs, executes steps 1, 2, 3, 13, and 14 of the breakpoint sequence, restores breakpoints, and resumes normal program execution.

When a breakpoint is placed on an IOT, EMT, TRAP, or any instruction causing a trap, ODT follows this algorithm:

1. When the breakpoint occurs as just described above, enters ODT.
2. When ;P is typed, sets the T-bit and executes the IOT, EMT, TRAP, or other trapping instruction.
3. Pushes the current PC and status (with the T-bit included) on the stack.
4. Obtains the new PC and status (no T-bit set) from the respective trap vector.
5. Executes the whole trap service routine without any breakpoints.
6. When an RTI is executed, restores the saved PC and PS (including the T-bit). Executes the instruction following the trap-causing instruction. If this instruction is not another trap-causing instruction, the T-bit trap occurs;

reinserts the breakpoints in the user program, or decrements the single-instruction mode repeat count. If the following instruction is a trap-causing instruction, repeats this sequence starting at step 3.

NOTE

You must use the RTI instruction to exit from the trap handler; otherwise, the T-bit is lost. ODT cannot regain control because the breakpoints have not yet been reinserted.

Note that the ;P command is invalid if a breakpoint has not occurred (ODT responds with ?). ;P is valid, however, after any trace trap entry.

The internal breakpoint status words have the following format:

- The first eight words contain the breakpoint addresses for breakpoints 0–7. (The ninth word contains the address of the next instruction to be executed in single-instruction mode.)
- The next eight words contain the respective repeat counts. (The following word contains the repeat count for single-instruction mode.)

You may change these words at will, either by using the breakpoint commands or by directly manipulating $B.

When program runaway occurs (when the program is no longer under ODT control, perhaps executing an unexpected part of the program where you did not place a breakpoint), give control to ODT by pressing the HALT key to stop the computer and then restarting ODT. The ODT asterisk prompts for another command.

If the program you are debugging uses the terminal for input or output, the program can interact with ODT to cause an error because ODT also uses the terminal. This interactive error does not occur when you run the program without ODT.

Note the following rules concerning the ODT break routine:

- If the console terminal interrupt is enabled upon entry to the ODT break routine, and no output interrupt is pending when ODT is entered, ODT generates an unexpected interrupt when returning control to the program.
- If the interrupt of the console terminal reader (the keyboard) is enabled upon entry to the ODT break routine, and the program is expecting to receive an interrupt to input a character, both the expected interrupt and the character are lost.
- If the console terminal reader (keyboard) has just read a character into the reader data buffer when the ODT break routine is entered, the expected character in the reader data buffer is lost.

Searches

The *word search* lets you search for bit patterns in specified sections of memory. Using the $M/ command, specify a mask, a lower search limit ($M+2), and an upper search limit ($M+4). Specify the search object in the search command itself.

The word search compares selected bits (where 1s appear in the mask) in the word and search object. If all of the selected bits are equal, the unmasked word displays.

The following shows the search algorithm.

1. Fetches a word at the current address.
2. XORs (exclusive OR) the word and search object.
3. ANDs the result of step 2 with the mask.
4. If the result of step 3 is zero, types the address of the unmasked word and its contents; otherwise, proceeds to step 5.
5. Adds two to the current address. If the current address is greater than the upper limit, types * and returns to the command decoder; otherwise, goes to step 1.

Note that if the mask is 0, ODT displays every word between the limits, since a match occurs every time (that is, the result of step 3 is always 0).

In the *effective address* search, ODT interprets every word in the search range as an instruction that is interrogated for a possible direct relationship to the search object. The mask register is opened only to gain access to the search limit registers.

The algorithm for the effective address search is as follows ((X) denotes contents of X, and K denotes the search object):

1. Fetches a word at the current address X.
2. If (X)=K [direct reference], displays contents and goes to step 5.
3. If (X)+X+2=K [indexed by PC], displays contents and goes to step 5.
4. If (X) is a relative branch to K, displays contents.
5. Adds 2 to the current address. If the current address is greater than the upper limit, performs a RET LF combination and returns to the command decoder; otherwise, goes to step 1.

Terminal Interrupt

When entering the TT SAVE routine, ODT follows these steps:

1. Saves the LSR status register (TKS).
2. Clears interrupt enable and maintenance bits in the TKS.
3. Saves the TT status register (TPS).

4. Clears interrupt enable and maintenance bits in the TPS.

To restore the TT:

1. Wait for completion of any I/O from ODT.
2. Restore the TKS.
3. Restore the TPS.

If the TT display interrupt is enabled upon entry to the ODT break routine, the following can occur:

- If no output interrupt is pending when ODT is entered, an additional interrupt always occurs when ODT returns control to the user.
- If an output interrupt is pending upon entry, the expected interrupt occurs when the user regains control.

If the TT reader (keyboard) is busy or done, the expected character in the reader data buffer is lost.

If the TT reader (keyboard) interrupt is enabled upon entry to the ODT break routine, and a character is pending, the interrupt (as well as the character) is lost.

Error Detection

ODT detects two types of error: invalid or unrecognizable command and bad breakpoint entry. ODT does not check for the validity of an address when you command it to open a location for examination or modification. In the following example, the command references nonexistent memory, causing a trap through the vector at location 4:

```
177774/
?MON-F-Trap to 4 003362
```

If the program you are debugging with ODT has requested traps through location 4 with the .TRPSET EMT, the program receives control at its TRPSET address.

If something other than a valid command is typed, ODT ignores the command and displays:

```
(echoes invalid command)?
*
```

ODT then prompts for another command. Therefore, to cause ODT to ignore a command that has just been typed, type any invalid character (such as 9 or RUBOUT), and the command will be treated as an error and ignored.

ODT suspends program execution whenever it encounters a breakpoint (that is, traps to its breakpoint routine). If the breakpoint routine is entered and no known breakpoint caused the entry, ODT displays:

```
BEnnnnnn
*
```

and prompts for another command. BEnnnnnn denotes bad entry from location nnnnnn. A bad entry may be caused by an invalid trace trap instruction, by a T-bit set in the status register, or by a jump to some random location within ODT.

Creating a Monitor-Independent ODT

The following procedure lets you create a monitor-independent ODT debugger; that is, an ODT that does not require that the operating system be loaded in memory.

The distributed debugger ODT.OBJ needs the operating system loaded in memory. You can modify ODT so it does not need the operating system loaded in memory, which could be useful, for example, in debugging the bootstrap.

Use the following procedure to create a debugger called ODTHWD.OBJ that functions independently of the operating system:

1. Use KED to create the following patch program. Name it ODTPAT.MAC. In the program, substitute for the symbol ..GVAL the value for that symbol located in the file CUSTOM.TXT:

   ```
   .TITLE  ODT
   .PSECT  $ODT$
   BASE=..GVAL-1000
   .=.+BASE
   BR      .+34
   .END
   ```

2. Assemble the created patch program:

   ```
   .R MACRO  RET
   *ODTPAT=ODTPAT  RET
   *  CTRL/C
   ```

3. Create the monitor-independent debugger ODTHWD.OBJ by modifying ODT.OBJ, which does not destroy the distributed ODT.OBJ but modifies a copy of it. Use the utility PAT.SAV in the following manner:

   ```
   .R PAT  RET
   *ODTHWD=ODT,ODTPAT  RET
   *  CTRL/C
   ```

4. Explicitly specify ODTHWD in the LINK/DEBUG command. If you do not specify ODTHWD, RT–11 by default links the distributed ODT:

   ```
   .LINK/DEBUG:ODTHWD program  RET
   ```

Chapter 19

Object-Module Patch Utility (PAT)

The RT–11 object-module patch (PAT) utility enables you to update code in a relocatable binary object module — OBJ file. PAT does not permit you to examine the octal contents of an object module. PAT makes the patch to the object module, using the procedure outlined in Figure 19–1. One advantage to using PAT is that you can add relatively large patches to an object module without performing any octal calculations. PAT accepts a file containing corrections or additional instructions and applies these corrections and additions to the original object module. You prepare correction input in source form and assemble it with the MACRO–11 assembler.

Two files form the input to PAT:

- Original input file

  The original input file consists of one or more concatenated object modules, only one of which can be corrected with a single execution of the PAT utility.

- Correction file

  The correction file contains the corrections and additions to that input file. The correction file consists of object code that, when linked by the linker, either replaces or appends to the original object module.

Output from PAT is the updated input file.

Always create a backup version of the file you want to patch before you use PAT to make the changes.

Calling PAT

To call PAT from the system device, respond to the dot (.) prompt displayed on the terminal by issuing the command:

```
.R PAT  RET
```

The Command String Interpreter (CSI) displays an asterisk at the left margin of the video terminal when it is ready to accept a command line. Chapter 1 describes the general syntax of the command line PAT accepts.

Press CTRL/C twice to halt PAT at any time (or press CTRL/C once to halt PAT when it is waiting for the terminal input) and return control to the monitor. To restart PAT, type R PAT in response to the monitor's prompt (.). When PAT completes an update operation it returns control to CSI level (*).

Using PAT

Figure 19–1 shows how PAT can update a file (FILE1) consisting of three object modules (MOD1, MOD2, and MOD3) by appending a correction file to MOD2. After running PAT, relink the updated module with the rest of the file to produce a corrected executable program.

Figure 19–1: Updating a Module, Using PAT

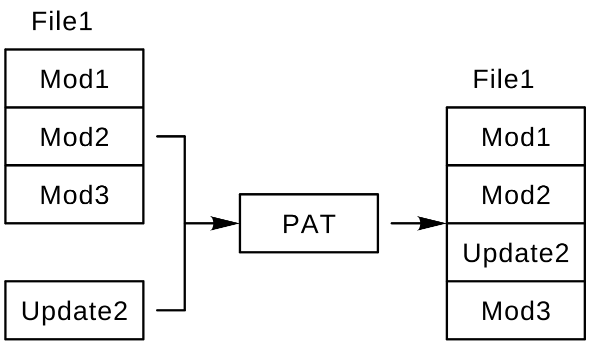
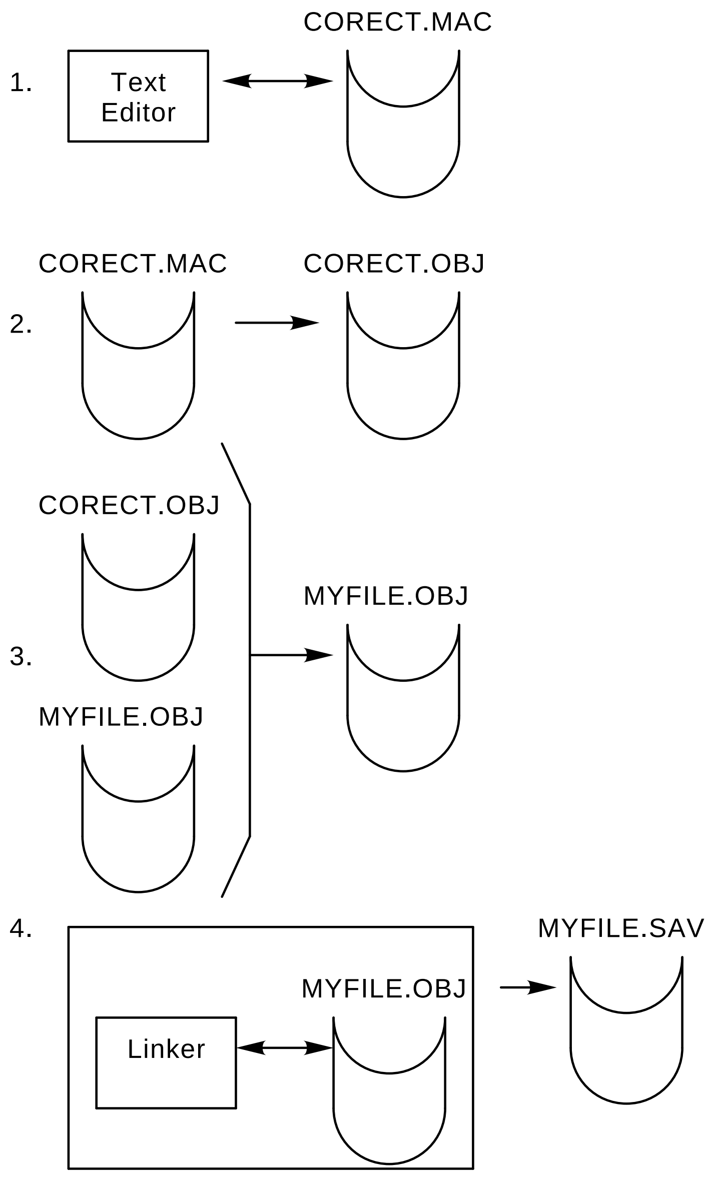


Figure 19–2 shows the four processing steps in generating an updated executable file, using PAT:

1. Create a correction file, using the text editor.
2. Execute the assembler (or compiler) to create an object module version of the file.
3. Execute PAT, using as input the correction file and the module to be updated.
4. This step varies with the object file being a program, an element in a library or an object module:
   - If the corrected object module is part of something that typically exists as a program (for example, BASIC), execute the linker to resolve new addresses and create an executable program.
   - If the corrected module is an element in a library (for example, SYSLIB), run the librarian and create or update the library to contain the new (corrected) object module.
   - If the corrected module typically exists as an object module (for example, ODT), you don’t have to do anything. Whenever you link this module, the corrections will be included.

Figure 19–2: Processing Steps Required to Update a Module, Using PAT



Command-Line Syntax

Specify the PAT command line in the following form:

[output-filespec]=input-filespec[/C[:n]],correct-filespec[/C[:n]]

where:

output-filespec	is the file specification for the output file. If you do not specify an output file, PAT does not generate one.
input-filespec	is the file specification for the input file. This file can contain one or more concatenated object modules.
correct-filespec	is the file specification for the correction file. This file contains the updates being made to a single module in the input file.
/C	specifies the checksum option for the associated file. This directs PAT to generate an octal value for the sum of all the binary data composing the module in that file. (See Section 21.5 for more information on checksums.)
n	specifies an octal value. PAT compares the checksum value it computes for a module with the octal value you specify.

No DCL Equivalents of PAT Utility Operations

The PAT utility is not accessible through DCL commands.

Updating Binary Code with PAT

PAT updates a base input module by using additions and corrections you supply in a correction file. This section describes the PAT input and correction files, and gives information on how to create the correction file.

Input Files

The input file is the file to be updated; it is the base for the output file and must be in object module format. When PAT executes, the module in the correction file applies to this file.

Correction File

The correction file must be in object module format and it is usually created from a MACRO–11 source file in the following format:

```
.TITLE  inputname
[.IDENT  updatenum]
[section  name]
inputline
inputline
   .
   .
   .
```

where:

inputname	is the name of the module to be corrected by the PAT update. That is, inputname must be the same name as the name on the input file .TITLE directive for a single module in the input file.
updatenum	is any value acceptable to the MACRO–11 assembler. Generally, this value reflects the update version of the file being processed by PAT, as shown in the examples below.
section name	is the ASECT, CSECT, or PSECT included in the correction file.
inputline	are lines of input PAT uses to correct and update the input file.

During execution, PAT adds to the module's symbol table any new global symbols defined in the correction file. Duplicate global symbols in the correction file will supersede their counterparts in the input file, provided that both definitions are relocatable or both are absolute.

A duplicate PSECT or CSECT will supersede the previous PSECT or CSECT, provided:

- Both have the same relocatability attribute (ABS or REL).
- Both are defined with the same directive (.PSECT or .CSECT).

If PAT encounters duplicate PSECT names, it sets the length attribute for the PSECT to the length of the longer PSECT and appends a new PSECT to the module.

If you specify a transfer address, it supersedes the transfer address of the module you are patching.

Updating Object Modules

The following examples show the source code for an input file and a correction file to be processed by PAT and the linker. The examples show as output a single source file that, if assembled and linked, would produce a binary module equivalent to the file generated by PAT and LINK. Two techniques are described:

- Overlaying lines in a module
- Appending a subroutine to a module

Overlaying Lines in a Module

In this example, PAT appends the correction file to the input file, then executes the linker to replace code within the input file.

The input file for this example is:

```
       .TITLE  ABC
       .IDENT  /01/
       .ENABL  GBL
ABC::
       MOV     A,C
       JSR     PC,XYZ
       RTS     PC
       .END
```

To add instruction ADD A,B after the JSR instruction, the following patch source file is included:

```
       .TITLE  ABC
       .IDENT  /01.01/
       .ENABL  GBL
.=.+12
       ADD     A,B
       RTS     PC
       .END
```

MACRO–11 assembles the patch source file (see example) and the resulting object file is input to PAT along with the original object file.

```
.TITLE  ABC
       .IDENT  /01.01/
       .ENABL  GBL
ABC::
       MOV     A,C
       JSR     PC,XYZ
       RTS     PC
.=ABC
.=.+12
       ADD     A,B
       RTS     PC
       .END
```

When the linker has processed these files, the load image appears, as shown in the following example:

```
       .TITLE  ABC
       .IDENT  /01.01/
       .ENABL  GBL
ABC::
       MOV     A,C
       JSR     PC,XYZ
       ADD     A,B
       RTS     PC
       .END
```

The linker uses the .=.+12 in the program counter field to determine where to begin overlaying instructions in the program and then overlays the RTS instruction with the patch code:

```
ADD  A,B
RTS  PC
```

Adding a Subroutine to a Module

Often a patch requires more than a few lines added to patch the file. A convenient technique adds new code by appending it as a subroutine at the end of the module, so that a JSR instruction is inserted to the subroutine at an appropriate location. The JSR directs the program to branch to the new code, execute that code, and then return to in-line processing.

The source code for the input file for the example is:

```
       .TITLE  ABC
       .IDENT  /01/
       .ENABL  GBL
ABC::
       MOV     A,B
       JSR     PC,XYZ
       MOV     C,R0
       RTS     PC
       .END
```

For example, suppose you wish to add the instructions:

```
MOV  D,R0
ASL  R0
```

between

```
MOV  A,B
```

and

```
JSR  PC,XYZ
```

The correction file to accomplish this is as follows:

```
        .TITLE  ABC
        .IDENT  /01.01/
        .ENABL  GBL
        JSR     PC,PATCH
        NOP
        .PSECT  PATCH
PATCH:
        MOV     A,B
        MOV     D,R0
        ASL     R0
        RTS     PC
        .END
```

PAT appends the correction file to the input file, then the linker processes the file, generating the output file:

```
        .TITLE  ABC
        .IDENT  /01.01/
        .ENABL  GBL
ABC::
        JSR     PC,PATCH
        NOP
        JSR     PC,XYZ
        MOV     C,R0
        RTS     PC
        .PSECT  PATCH
PATCH:
        MOV     A,B
        MOV     D,R0
        ASL     R0
        RTS     PC
        .END
```

In this example, the JSR PC,PATCH and NOP instructions overlay the three-word MOV A,B instruction. (The NOP is included because this is a case where a two-word instruction replaces a three-word instruction. NOP is required to maintain alignment.) The linker allocates additional storage for .PSECT PATCH, writes the specified code into this program section, and binds the JSR instruction to the first address in this section. (Note that the MOV A,B instruction, replaced by the JSR PC,PATCH, is the first instruction the PATCH subroutine executes.)

Determining and Validating the Contents of a File

Use the checksum option (/C) to determine (validate) the contents of a module. The checksum option directs PAT to compute the sum of all binary data composing a file. Specifying the command in the form /C:n, /C, directs PAT to compute the checksum and compare that checksum to the value you specify as *n*.

To determine the checksum of a file, enter the PAT command line with the /C option applied to the appropriate file (the file whose checksum you want to determine). For example, PAT responds to the command:

```
=INFILE/C,INFILE.PAT
```

with the message:

```
?PAT-W-Input module checksum is nnnnnn
```

PAT generates a similar message when you request the checksum for the correction file.

To validate the changes made to a file, enter the checksum option in the form /C:n. PAT compares the value it computes for the checksum with the value you specify as n. If the two values do not match, PAT enters the changes but displays a message reporting the checksum error as either:

```
?PAT-W-Input file checksum error
```

or

```
?PAT-W-Correction file checksum error
```

NOTE

Checksum processing always results in a non-zero value. Do not confuse this checksum with the record checksum byte.

Chapter 20

Peripheral Interchange Utility (PIP)

The Peripheral Interchange Utility (PIP) is a file-transfer and file-maintenance program. You can use PIP to transfer files between any supported RT–11 devices and to merge, rename, delete, and change the protection status of files.

Calling and Terminating PIP

To call PIP from the system device, respond to the keyboard monitor prompt (.) by typing:

```
.R PIP  RET
```

The Command String Interpreter (CSI) displays an asterisk at the left margin of the terminal and waits for you to type a command string. If you only press RETURN at this point, PIP displays its current version number and prompts you again for a command string. You can type CTRL/C to halt PIP and return control to the monitor when PIP is waiting for input from the terminal. You must type CTRL/C twice to abort PIP at any other time. To restart PIP, type R PIP or REENTER and press RETURN in response to the monitor's dot.

PIP Command-Line Syntax

Chapter 1 of the *RT–11 System Utilities Manual*, Part I, describes the general syntax of the command line PIP accepts. You can specify as many as six input files, but only one output file.

Because PIP performs file transfers for all RT–11 data formats (ASCII, object, and image), it does not assume file types for either input or output files. You must explicitly specify all file types where file types are applicable.

Entering a Date Argument to a Command

Some of the PIP options accept a date as an argument. The syntax for specifying the date is:

[:dd.][:mmm][:yy.]

where:

dd.	specifies the day (a decimal integer in the range 1–31).
mmm	specifies the first three characters of the name of the month.
yy.	specifies the year (a decimal integer in the range 73–99).

The default value for the date is the current system date. If you omit any of the date values (dd, mmm, or yy), RT–11 uses the values from the current system date. For example, if you specify only the year ::90. and the current system date is May 4, l990, RT–11 uses the date 4.:MAY:90.. If the current date is not set, it is considered 0 (the same as for an undated file in a directory listing).

On random-access devices such as disks, and in transfers from magtape, PIP operations retain a file's creation date. If the file's creation date is 0, PIP gives it the current system date. However, in transfers to magtape, PIP always gives files the current system date.

Getting a –BAD– File-Creation Date

If you have selected timer support through the system-generation process, but have not selected automatic end-of-month date advancement, make sure that you set the date at the beginning of each month with the DATE command. If you fail to set the date at the beginning of each month, the system displays –BAD– in the creation date column of each file created beyond the end-of-month.

Note that you can eliminate –BAD– by using the RENAME/SETDATE command after you set the date.

Changing the Dates, Protection, and Names of Files

If you specify a command involving random-access devices for which the output specification is the same as the input specification, PIP does not move any files. However, it can change the creation dates on the files if you use /T, it can rename the files if you use /R, it can protect files if you use /F, or it can remove protection from files if you use /Z.

Using Wildcards with PIP

You can use all variations of wildcards for the input-file specifications in the PIP command line. However, you cannot use embedded wildcards in output-file specifications. If you use a wildcard in an input-file specification, the corresponding output-file name or file type must be an asterisk. (The concatenate copy operation is an exception to this rule because it does not allow wildcards in the output specification.) following gives you the standard subhead one)

The Order in Which PIP Operates

In most cases, PIP performs operations on files in the order in which they appear in the device directory. In transfers from magtape (and for all other transfers requested on the same command line), PIP performs operations on files in the order in which they appear on the volume.

Files PIP Ignores

When you use wildcards in an input file type, PIP ignores system files with the file type SYS unless you also use the /Y option. PIP displays the message *?PIP–W–No SYS action* if you omit the /Y option on a command that would operate on SYS files.

PIP ignores all files with the file type BAD unless you explicitly specify both the file name and file type in the command string. PIP does not display a warning message when it does not include BAD files in an operation.

This example transfers all files, including system files (regardless of file name or file type), from device DK to device DU1. It does not transfer BAD files:

```
*DU1:*.*/Y=*.*
```

NOTE

You cannot perform any operations that result in deleting a protected file. For example, you cannot transfer a file to a volume if a protected file with the same name already exists on the output volume.

Examples

1. In the following example, the embedded percent character (%) specifies a valid file-name character and the asterisk specifies a valid file name:

   ```
   **.B=A%B.MAC
   ```

2. The next command deletes all files with the file type BAK (regardless of their file names) from device DK:

   ```
   **.BAK/D
   ```

3. This command renames all files with a BAK file type (regardless of file names) so that these files now have a TST file type (maintaining the same file names):

   ```
   **.TST=*.BAK/R
   ```

PIP Option Summary

The options you can use with PIP are summarized in Table 20–1. If you do not specify an option, PIP assumes that the operation is a file transfer in image mode.

You can put command options at the end of the command string or type them after any file name in the string. Operations involving magtape are an exception, because the /M option is device dependent and has a different meaning when you specify it on the input or output side of a command line.

You can type any number of nonconflicting options in a command line. For example, you can combine copy and delete operations in one line. You can also combine the protect and noprotect options with copy and rename operations.

Table 20–1: PIP Option Summary

Option	Function
/A	Copies files in ASCII mode, ignoring and discarding nulls and rubouts. It converts input file to 7-bit ASCII and treats CTRL/Z (32 octal) as the logical end-of-file on input (the default copy mode is image).
/B	Copies files in formatted binary mode (the default copy mode is image).
/C[:date]	Used with other options to include only files with the specified date in the operation. If you use /C and do not specify a date, PIP includes only files with the current date in the specified operation.
/D	Deletes input files from a specific device. Note that PIP does not automatically query before it performs the operation. If you combine /D with a copy operation, PIP performs the delete operation after the copy completes. This option is invalid in an input specification with magtape.
/E	Transfers files in a single- or small-disk system. PIP initiates the transfer, but pauses and waits for you to mount the volumes involved in the transfer.
/F	Protects files from deletion. Gives protected status to output files during a copy operation so you cannot delete them. If you use neither /F nor /Z, the output files retain the protection status of the input files. Can also be used with /R. Invalid for magtapes.
/G	Ignores any input errors that occur during a file transfer and continues copying.
/H	Verifies that the output file matches the input file after a copy operation. Cannot be used with /A or /B.
/I[:date]	Used with other options to include only files created on or after the specified date.
/J[:date]	Used with other options to include only files created before the specified date.
/K:n	Makes n copies of the output files to any sequential device, such as LP, or TT.
/M:n	Used when I/O transfers involve magtape.

Table 20–1 (Cont.): PIP Option Summary

Option	Function
/N	Does not copy or rename a file if a file with the same name exists on the output device. This option protects you from accidentally deleting a file. It is invalid for magtape in the output specification.
/O	Deletes a file on the output device if you copy a file with the same name to that device. The delete operation occurs before the copy operation. This option is invalid for magtape in the output specification.
/P	Copies or deletes all files except those you specify.
/Q	Use only with another operation. The /Q option causes PIP to display the name of each file to be included in the operation you specify. You must respond with a Y to include a particular file.
/R	Renames the file you specify. This operation is invalid for magtape.
/S	Copies files one block at a time.
/T[:date]	Puts the specified date on all files involved in the operation. This option is invalid when copying to magtape; operations involving magtape devices always use the current date.
/U	Copies and concatenates all files you specify.
/V	Copies files from one input volume to two or more smaller output volumes.
/W	Displays on the terminal a log of all files involved in the operation.
/X	Causes PIP to display an information message instead of a fatal message when it cannot find a file you specified in the command line.
/Y	Includes SYS files in the operation you specify. You cannot modify or delete these files unless you use the /Y option when you use wildcards in the input file types.
/Z	Removes protected status from output files so you can delete them. If you use neither /F nor /Z, the output files retain the protection status of the input files. When used with /R, enables files for deletion if they have been previously protected with /F. Invalid for magtapes.

PIP Option Descriptions

Operations Involving Magtape (/M:n)

PIP handles magtape, which is a sequential-access device, differently from random-access devices, such as disks, diskettes, and DECtape II. On magtape, files are stored serially, one after another, and there is no directory at the beginning of each device that lists the files and gives their location. Thus, you can access only one file at a time on each sequential-access device unit. Avoid commands that specify the same device-unit number for both the input and output files—they are invalid.

The /M:n option makes operations that involve magtape more efficient. This option lets you specify different tape handling procedures for PIP to follow. The following sections outline the operations that involve magtape and describe the different procedures for using these devices that you can specify with the /M:n option. Remember that when you use the /M:n option, n is interpreted as an octal number. You must use n. (n followed by a decimal point) to specify a decimal number.

Magnetic tape is a convenient auxiliary storage medium for large amounts of data, and is often used as backup for disks. Reflective strips indicate the beginning and end of the tape. A special label (an EOF1 or EOV1 tape label) followed by two tape marks indicates the end of current data and also where new data can begin.

Valid Options to Use with Magtape

The following PIP options are valid for use with magtape: /A, /B, /C[:date], /F, /G, /H, /I, /J, /M, /P, /Q, /S, /U, /V (only when magtape is the output volume), /W, /X, /Y, and /Z. These options are invalid with magtape: /E, /K, /R, /T, and /V (when magtape is the input volume). The /M:n option lets you direct the tape operation; you can move the tape and perform an operation at the point you specify. Note that /D is invalid for input from magtape; /N and /O are invalid for output to magtape.

/M:n Is Postion Dependent

The /M:n option can be different for the output and input side of the command line. Since the option applies to the device and not to the files, you can specify one /M:n option for the output file and one for each input file.

The Procedure for Locating a Magtape File

Sometimes PIP begins an operation at the current position. To determine the current position, the magtape handler backspaces from its present position on the tape until it finds either an EOF indicator or the beginning of tape (BOT), whichever comes first. PIP then begins the operation with the file that immediately follows the EOF or BOT. The magtape handler also has a special procedure for locating a file with sequence number n:

1. If the file sequence number is greater than the current position, PIP searches the tape in the forward direction.
2. If the file sequence number is more than one file before the current position, or if the file sequence number is less than five files from BOT, the tape rewinds before PIP begins its search.
3. If the file sequence number is at the current position, or if it is one file past the current position, PIP searches the tape in the reverse direction.

Whenever you fetch or load a new copy of the magtape handler, the tape position information is lost. The new handler searches backward until it locates either BOT or a label from which it can learn the position of the tape. It then operates normally, according to steps 1, 2, and 3 described above.

If you omit the /M:n option, the tape rewinds between each operation. Using /M:0 has the same effect as omitting /M:n. When n is positive, it specifies the file sequence number. When n is negative, it specifies an instruction to the magtape handler.

Copying from Magtapes

In copying from magtapes, /M:n functions as follows:

1. If n is 0:

   The tape rewinds and PIP searches for the file you specify. If you specify more than one file, the tape rewinds before each search. If the file specification contains a wildcard, the tape rewinds only once and then PIP copies all the appropriate files.

2. If n is a positive integer:

   PIP goes to file sequence number n. If the file it finds there is the one you specified, PIP copies it. Otherwise, PIP displays the *?PIP–F–File not found DEV:FILNAM.TYP* message. If you use a wildcard in the file specification, PIP goes to the file sequence number n and then begins to search for matching files.

3. If n is -1:

   PIP starts the search at the current position. If the current position is not the beginning of the tape, PIP may not find the file you specify, even though it does exist on the tape.

Writing to Magtapes

In writing to magtapes, /M:n functions as follows:

1. If n is 0:

   The tape rewinds before PIP copies each file. PIP displays a warning message if it finds a file with the same name and file type as the input file and does not perform the copy operation.

2. If n is a positive integer:

   PIP goes to the file sequence number n and enters the file you specify. If PIP reaches logical end-of-tape (LEOT) before it finds file sequence number n, it displays the *?PIP–F–File sequence number not found* message. If you specify more than one file or if you use a wildcard in the file specification, the tape does not rewind before PIP writes each file, and PIP does not check for duplicate file names.

3. If n is -1:

   PIP goes to the LEOT and enters the file you specify. It does not rewind, and it does not check for duplicate file names.

4. If n is -2:

   The tape rewinds between each copy operation. PIP enters the file at LEOT or at the first occurrence of a duplicate file name.

Writing a Logical End-of-Tape

If PIP reaches the physical end-of-tape before it completes a copy operation, it cannot continue the file on another tape volume. Instead, it deletes the partial file by backspacing and writing a logical end-of-tape over the file's header label. You must restart the operation and use another magtape.

If you type CTRL/C twice during any output operation to magtape, PIP does not write a logical end-of-tape at the end of the data. Consequently, you cannot transfer any more data to the tape unless you follow one of the following recovery procedures.

1. Transfer all good files from the interrupted tape to another tape and initialize the interrupted tape in the following manner:

```
*MU1:*.*=MU0:*.*
_<CTRL/C>
.R DUP
*MU0:/Z/Y
```

2. Determine the sequential number of the file that was interrupted and use the /M:n construction to enter a replacement file (either a new file or a dummy) over the interrupted file. PIP writes the replacement file and a good LEOT after it. The following example assumes the bad file is the fourth file on the tape:

```
*MU0:file.new/M:4=file.dum
```

Copy Operations

PIP copies files in image, ASCII, and binary format. Other options let you change the date on the files, access SYS files, combine files, change a file's protection status, and perform other similar operations. PIP automatically allocates the correct amount of space for new files in copy operations. For block-replaceable devices, PIP stores the new file in the first empty space large enough to accommodate it. If an error occurs during a copy operation, PIP displays a warning message, stops the copy operation, and prompts you for another command. You cannot copy BAD files unless you specifically type each file name and file type.

Copying Files in Image Mode (the Default)

If you enter a command line without an option, PIP copies files onto the destination device in image mode. Note that you cannot reliably transfer memory image files to the printer or terminal. PIP can image-copy ASCII and binary data but it does not do any of the data checking described in Section 13.4.2.3.

The following command makes a copy of the file named XYZ.SAV on device DK and assigns it the name ABC.SAV. (Both files exist on device DK after the operation.):

```
*ABC.SAV=XYZ.SAV
```

The next example copies from DK all MAC files whose names are three characters long and begin with A. PIP stores the resulting files on DY1:

```
*DY1:*.*=A%%.MAC
```

Copying Files in ASCII Mode (/A)

Use the /A option to copy files in 7-bit ASCII mode. PIP ignores and eliminates nulls and rubouts during file transfer. PIP treats CTRL/Z (32 octal) as logical end-of-file if it encounters that character in the input file. You cannot use the /A option with the /V option.

The following command copies F2.FOR from device DK onto device DY1 in ASCII mode and assigns it the name F1.FOR:

```
*DY1:F1.FOR=F2.FOR/A
```

Copying Files in Binary Mode (/B)

Use the /B option to transfer formatted binary files (such as OBJ files produced by the assembler or the FORTRAN compiler and LDA files produced by the linker). You cannot use the /B option with the /V option.

The following command transfers a formatted binary file from device DL0: to device DK and assigns it the name FILE.OBJ:

```
*DK:FILE.OBJ=DL:F3.OBJ/B
```

When performing formatted binary transfers, PIP displays a warning if a checksum error occurs. If there is a checksum error and you did not use /G to ignore the error, PIP does not perform the copy operation. You cannot copy library files with the /B option. Copy library files in image mode.

Specifying a Date (/C[:date])

The /C[:date] option includes only those files with the specified date. If no date is specified only those files with the current date are included. Specify /C only once in the command line; it applies to all the file specifications in the entire command.

The following command copies (in ASCII mode) all files with the file type MAC on DL0 that also have the date January 12, l991. It also copies the file RDWR.MAC, if it has the date January 12, l991, from DY0 to DY1. It combines all these files under the name NN3.MAC on DY1:

```
*DY1:NN3.MAC=DL0:*.MAC/C:12.:JAN:91.,DY0:RDWR.MAC/A/U
```

The next command copies all files with the current date (except SYS and BAD files) from DK to DY1. This is an efficient way to back up all new files after a session at the computer:

```
*DY1:*.*=*.*/C
```

Deleting Files (/D)

Use the /D option to delete one or more files from the device you specify. Note that PIP does not query you before it performs this operation unless you use /Q. Remember to use the /Y option to delete SYS files if you use wildcards in the input file types. You cannot delete BAD files, unless you name each one specifically, including file name and file type. You can specify only six files in a delete operation unless you use wildcards. You must always indicate a file specification in the command line. A delete command consisting only of a device name (dev:/D) is invalid. The delete option is also invalid for magtape.

The following examples illustrate the delete operation:

```
*FILE1.SAV/D
```

The command shown above deletes FILE1.SAV from device DK:

```
*DY1:*.*/D
?PIP-W-No .SYS action
*
```

The command shown above deletes all files from device DY1: except those with a SYS or BAD file type. Since there is a file with a SYS file type, PIP displays a warning message to remind you that this file has not been deleted:

```
**.MAC/D
```

This command deletes all files with a .MAC file type from device DK:

Initiating a Copy Operation and Waiting (/E)

If you have a single-disk system or a diskette system, you will find the /E option useful for copy operations. Use this option when you need to change storage volumes during a copy procedure. The following is the general format of the command line:

filespec/E=filespec

You can use any option with /E that is valid with your RT–11 configuration. You cannot use wildcards as input. When you use /E, make sure that PIP is on your system volume.

The Procedure for Using the /E Option

When you use the /E option, PIP guides you through a series of steps in the process of completing the file transfer.

1. PIP initiates execution of the command, but then pauses and displays the message:

   ```
   Mount input volume in <device>; Continue?
   ```

   where the <device> specifies the device in which you mount the input volume.
2. At this time you can remove the system volume (if necessary) and mount the volume on which you actually want the operation to take place.
3. When the new volume is loaded, type Y or any string beginning with Y followed by a RETURN to execute the operation. If you type N or any string beginning with N, or CTRL/C, the operation is not completed. Instead PIP prompts you to remount the system volume if you have removed it and the monitor prompt (.) appears. Any other response causes the message to repeat.
4. If you type Y, PIP prompts you for the input volume, if any. When the operation completes, the following message is displayed:

   ```
   Mount system volume in <device>; Continue?
   ```
5. Replace the system device and type Y or any string beginning with Y followed by a RETURN. If you type any other response, PIP prompts you to mount the system volume until you type Y. When you type Y, the asterisk (*) prompt is displayed, and PIP waits for you to enter another command.

The sections that follow describe the procedures for single-drive and double-drive transfer.

Transferring Files Using One Device Drive

If you want to transfer a file between two storage volumes, and you have only one drive for that type of storage volume, use the following procedure:

1. Enter a command string according to this general syntax:

   output-filespec/E=input-filespec

   where output-filespec specifies the destination device and file specification, and input-filespec specifies the source device and file specification.
2. PIP responds by displaying the following message at the terminal:

   ```
   Mount input volume in <device>; Continue?
   ```

   where <device> specifies the device into which you are to mount your input volume. Type Y or any string beginning with Y followed by a RETURN after you have mounted your input volume. If you type any string beginning with N or if you type CTRL/C, the operation is not performed and the monitor prompt (.) appears. If you have removed the system volume, PIP prompts you to remount it.

3. PIP continues the copy procedure and displays the following message on the terminal:

   ```
   Mount output volume in <device>; Continue?
   ```

   After you have removed your input volume from the device, mount your output volume and type Y or any string beginning with Y followed by RETURN. If you type any string beginning with N or if you type CTRL/C, the operation is not performed and the monitor prompt (.) appears. If you have removed the system volume, PIP prompts you to remount it.

4. Depending on the size of the file, PIP may repeat the transfer cycle (steps 2 and 3) several times before the transfer is complete. When the transfer is complete, PIP displays the following message if you had to remove the system volume from <device>:

   ```
   Mount system volume in <device>;  Continue?
   ```

   When you remount the system volume and type Y or any string beginning with Y followed by a RETURN in response to the last instruction, you complete the copy operation. If you type anything other than Y, PIP continues to prompt you to remount the system volume until you type Y.

Transferring Files Using a Double Device Drive

You can use the /E option for transferring files between two nonsystem volumes. The following is the procedure for transferring files this way:

1. With your system volume mounted, enter a command string according to the following general syntax:

   output-filespec/E=input-filespec

   where output-filespec specifies the destination device and file specification, and input-filespec specifies the source device and file specification.

2. After you have entered the command string, PIP responds with the message:

   ```
   Mount input volume in <device>; Continue?
   ```

   Type Y or any string beginning with Y followed by a RETURN when you have mounted the input volume. If you type any string beginning with N or if you type CTRL/C, the operation is not performed and the monitor prompt (.) appears. If you have removed the system volume, PIP prompts you to remount it.

3. PIP then displays:

   ```
   Mount output volume in <device>; Continue?
   ```

   Type Y or any string beginning with Y followed by a RETURN after you have mounted the output volume. If you type any string beginning with N or if you type CTRL/C, the operation is not performed and the monitor prompt (.) appears. If you have removed the system volume, PIP prompts you to remount it.

4. Unlike the single-volume transfer, the double-volume transfer involves only one cycle of mounting the input and output volumes. When the file transfer is complete, PIP displays the following message if you had to remove the system volume from <device>:

   ```
   Mount system volume in <device>; Continue?
   ```

   When you type Y or any string beginning with Y followed by a RETURN in response to the last instruction, you complete the copy operation. If you type anything other than Y, PIP continues to prompt you to mount the system volume until you type Y.

Protecting Files (/F)

Use the /F option to protect files. The letter P next to the block size number in the file's directory entry indicates the file is protected.

If a file is protected you cannot perform any operations on it that result in deleting the file. You can copy a protected file to another volume or change its name. However, you cannot change its protected status unless you use the /Z (no protection) option. Note that the contents of a protected file are not protected; that is, although you cannot delete a protected file, you can change or delete its contents.

You can also use the /F option during copy operations to protect the output file, and with /R to change a file's protection status. If during a copy operation you use neither /F nor /Z, the output files retain the protection status of the input files.

The following command protects all files with the file type MAC on DK:

```
**.MAC/F
```

The following command copies all files with file type ORI from DL0 to DL1. The resulting output files on DL1 are protected from deletion:

```
*DL1:*.*=DL0:*.ORI/F
```

If you use the /F option with a file that is already protected, no operation is performed on that file regardless of any other options in the command string. For example, the following command requests PIP to protect the file DY1:CALCAB.MAC and change its creation date to April 21, 1991. However, because the file is already protected, PIP performs neither operation:

```
*DL1:CALCAB.MAC/F/T:21.:APR:91.
```

Ignoring Input Errors (/G)

The /G option copies files, but ignores all input errors. This option forces a single-block transfer, which you can invoke at any other time with the /S option. Use the /G option if an input error occurred when you tried to perform a normal copy operation. The procedure can sometimes recover a file that is otherwise unreadable. If an error still occurs, PIP displays the *?PIP–W–Input error DEV:FILNAM.TYP* message and continues the copy operation.

The following command copies the file TOP.SAV in image mode from device DY1: to device DK and assigns it the name ABC.SAV:

```
*ABC.SAV=DY1:TOP.SAV/G
```

The next command copies files F1.MAC and F2.MAC in ASCII mode from device DY0 to device DY1. This command creates one file with the name COMB.MAC, and ignores any errors that occur during the operation:

```
*DY1:COMB.MAC=DY0:F1.MAC,F2.MAC/A/G/U
```

Verifying Your Work (/H)

Use the /H option to verify that the output file matches the input file when a copy operation is performed. If the two files are different a message is displayed on the terminal. This option cannot be used with /A or /B.

The following command verifies that the output file A.BAK on DY1 is the same as the input file A.MAC on DY0:

```
*DY1:A.BAK=DY0:A.MAC/H
```

Selecting Since Files Only (/I[:date])

The /I[:date] option includes only those files created on or after the specified date. If no date is specified, PIP uses the current date.

The following command copies from DK only those MAC files created on or after January 4, l991:

```
*DL0:*.MAC=*.MAC/I:4.:JAN:91.
```

Selecting Before Files Only (/J[:date])

The /J[:date] option includes only those files created before the specified date. If you do not specify a date, PIP uses the current date.

The following command copies only those MAC files created before January 14, l991:

```
*DL0:*.MAC=*.MAC/J:14.:JAN:91.
```

Making Several Copies (/K:n)

The /K:n option directs PIP to generate n copies of the file you specify. The only valid output devices are the console terminal and the printer. Normally, each copy of the file begins at the top of a page; copies are separated by form feeds:

```
*LP:=STOTLE.LST/K:3
```

This command, for example, prints three copies of the listing file, STOTLE.LST, on the printer.

Preventing a Replace Operation (/N)

The /N option prevents execution of a copy or rename operation if a file with the same name as the output file already exists on the output device. This option is not valid when output is to magtape.

The following example uses the /N option:

```
*DY0:CT.SYS=DK:CT.SYS/Y/N
?PIP-W-Output file found, no operation performed DK:CT.SYS
*
```

The file named CT.SYS already exists on DY0, and the copy operation does not proceed.

Predeleting Files (/O)

The /O option deletes a file on the output device if you copy a file with the same name to that device. PIP deletes the file on the output device before the copy operation occurs. Normally, PIP deletes a file of the same name after the copy completes. This option is not valid when output is to magtape.

The following example uses the /O option:

```
*DL1:TEST1.MAC=DY1:TEST.MAC/O
```

If a file named TEST1.MAC already exists on DL1, PIP deletes it before copying TEST.MAC from DY1 to TEST1.MAC on DL1:

Excluding Files from the Operation (/P)

The /P option directs PIP to include all files in the operation except the ones you specify. Note that if you want to include system (SYS) files and you use the /P option, you must always use the /Y option with it:

```
*DY0:*.*=DY1:*.MAC/P
```

This command directs PIP to transfer all files from DY1 to DY0 except the .MAC files. The SYS files will also be excluded from the operation because the /Y option was not specified.

Requesting Confirmation (/Q)

Use the /Q option with another PIP operation to list all files and to request confirmation for each file before it is included. Typing Y or any string beginning with Y followed by a RETURN causes the named file to be processed; typing anything else excludes the file.

The following example deletes four files from DY1:

```
*DY1:*.*/D/Q
 Files deleted:
DY1:FIX463.SAV?
DY1:GRAPH.BAK ? Y
DY1:DMPX.MAC  ?
DY1:MATCH.BAS ?
DY1:EXAMP.FOR ?
DY1:GRAPH.FOR ? Y
DY1:GLOBAL.MAC? Y
DY1:PROSEC.MAC? Y
DY1:KB.MAC    ?
DY1:EXAMP.MAC ?
*
```

Renaming Files (/R)

Use the /R option to rename a file you specify as input, giving it the name you specify in the output specification. The input and output volumes for a rename operation must be the same. PIP displays an error message if the command specifications are not valid. Use the /Y option if you rename SYS files and you use wildcards in the input file types. You cannot use /R with magtape.

The following examples illustrate the rename operation:

```
*DY1:F1.MAC=DY1:F0.MAC/R
```

The command shown above renames F0.MAC to F1.MAC on device DY1:

```
*DL1:OUT.SYS=DL1:CT.SYS/R
```

This command renames file CT.SYS to OUT.SYS.

The rename command is particularly useful when a file contains bad blocks. By giving the file a BAD file type, you can ensure that the file permanently resides in that area of the device. Thus, the system makes no other attempts to use the bad area. Once you give a file a BAD file type, you cannot move it during a compress operation. You cannot rename BAD files unless you specifically indicate both the file name and file type.

Copying Files One Block at a Time (/S)

The /S (Single-Block) option directs PIP to copy files one block at a time. On some devices, this operation increases the chances of an error-free transfer. You can combine the /S option with other PIP copy options. For example:

```
*DL1:TEST.MAC=DL0:TEST.MAC/S
```

PIP performs this transfer one block at a time.

Placing a Creation Date on Files (/T[:date])

This option causes PIP to put the specified date on all files involved in the operation. If you specify no date, PIP uses the current system date. Normally, PIP preserves the existing file creation date on copy and rename operations. This option is invalid when copying to magtape, because PIP always uses the current date for these operations.

The following command copies all the files with file type .COM copied from DY0 to DY1, and assigns the output files the date January 24, l991:

```
*DY1:*.*=DY0:*.COM/Y/T:24.:JAN:91.
```

Combining Several Files into One File (/U)

To combine more than one file into a single file, use the /U option. This option is particularly useful when you want to combine several object modules into a single file for use by the linker or librarian. PIP does not accept wildcards on the output specification. Use the /B option with /U if you are concatenating object (.OBJ) files.

The following examples show the /U option:

```
*DK:AA.OBJ=DY1:BB.OBJ,CC.OBJ,DD.OBJ/U/B
```

The command shown above transfers files BB.OBJ, CC.OBJ, and DD.OBJ to device DK as one file and assigns it the name AA.OBJ:

```
*DL1:MERGE.MAC=DL0:FILE2.MAC,FILE3.MAC/A/U
```

This command merges ASCII files FILE2.MAC and FILE3.MAC on DL0 into one ASCII file named MERGE.MAC on device DL1.

Copying Files from One to Several Volumes (/V)

The /V (Multivolume) option copies files from an input volume to two or more smaller output volumes. This option is useful when you are copying several files from a large input volume and you are not sure whether all the files will fit on one output volume.

When you use this option PIP copies files to the output volume until the system finds a file that will not fit. PIP continues to search that file’s directory segment, copying all files from the segment that will fit onto the output volume. When no more files from that segment will fit on the output volume, PIP prompts you to mount the next output volume and displays the Continue? message. Mount another output volume of the same type and type Y or any string beginning with Y followed by a RETURN to continue the copy operation. If you type any string beginning with N or if you type CTRL/C, the operation is aborted and the monitor prompt (.) appears.

When you type Y to continue, PIP copies the first file that would not fit to the previous output volume to the new output volume. PIP continues to copy files from that directory segment until no more files from that segment will fit on the output volume or until all files from that directory segment have been copied. If all files from that segment have been copied, PIP begins copying files from the next directory segment. File copying continues in this fashion until all the specified input files have been copied.

The following example copies all files on DL0 to several double-density diskettes:

```
*DY0:*.*=DL0:*.*/V
Mount next output volume in DY0:; Continue? Y
Mount next output volume in DY0:; Continue? Y
Mount next output volume in DY0:; Continue? Y
*
```

Logging Your Work (/W)

When you use the /W option, PIP displays a list of all files included in the operation. The /W option is useful if you do not want to take the time to use the query mode (the /Q option description), but you do want a list of the files operated on by PIP.

PIP displays the log for an operation on the terminal under the command line. This example shows logging with the delete operation:

```
*DY1:*.*/D/W
?PIP-W-No .SYS action
 Files deleted:
DY1:TEST.MAC
DY1:FIX463.SAV
DY1:GRAPH.BAK
DY1:DMPX.MAC
DY1:MATCH.BAS
DY1:EXAMP.FOR
DY1:GRAPH.FOR
DY1:GLOBAL.MAC
DY1:PROSEC.MAC
DY1:EXAMP.MAC
*
```

Ensuring Operation Completes (/X)

The /X option causes PIP to display an information message when PIP fails to find all of the files you specify in a command line. If you do not use the /X option, PIP displays a fatal error message when it is unable to find an input file, and control returns to the keyboard monitor after the operation completes. Use /X in indirect command files to ensure that processing will continue even if PIP fails to find a file you specify.

In the following example, the input files FILE1.TXT and FILE3.TXT are copied to DL1:. However, since the system is unable to find DL0:FILE2.TXT, PIP displays a message to inform you:

```
*DL1:*.*=DL0:FILE1.TXT,FILE2.TXT,FILE3.TXT
?PIP-I-File not found DL0:FILE2.TXT
```

Enabling Operations on System Files (/Y)

Use the /Y option if you need to perform an operation on system (SYS) files and you use wildcards in the input file type. For example:

```
**.*=DY1:*.*/Y
```

This command copies to device DK, in image mode, all files (including SYS files) from device DY1:. Note that you must always use /Y with the /P option to include SYS files, even when you use no wildcards.

Removing the Protection Status from Files (/Z)

Use the /Z option to remove protected status from files, so that you can delete or change those files. You can also use the /Z option with /R to change the protection status of a file, and during copy operations to remove protection from the output file.

Note that since you cannot delete files assigned as logical disks, you cannot use the /Z option to remove protection from these files.

The following command removes protection from all MAC files on DK:

```
**.MAC/Z
```

The following command copies the file PROGRM.MAC from DY0 to DY1. The resulting output file on DY1 is enabled for deletion:

```
*DY1:PROGRM.MAC=DY0:PROGRM.MAC/Z
```

If you use the /R option with a file that is already unprotected, no operation is performed on that file regardless of any other options in the command string. For example, the following command requests PIP to unprotect the file DY1:CALCAB.MAC and change its creation date to April 21, 1990. However, because the file is already unprotected, PIP performs neither operation:

```
*DL1:CALCAB.MAC/R/T:21.:APR:90.
```

DCL Equivalents of PIP Utility Operations

Table 20–2 lists the DCL commands that are equivalent to PIP utility operations.

Seven different DCL commands use PIP: COPY, DELETE, PRINT, PROTECT, RENAME, TYPE, and UNPROTECT. Those commands that can be used with a PIP option are placed in the column next to that PIP option; and those DCL options that can be used with those commands are placed in the next column beside the commands. For example, the PIP /A option is equivalent to the command COPY /ASCII. The extended hyphen (–) in the option column means there are no DCL options for the command(s) on the same line that use PIP.

Table 20–2: DCL Equivalents of PIP Utility Operations

CSI Option	DCL Command(s)	DCL Option(s)
/A	COPY	/ASCII
/B	COPY	/BINARY
/C[:date]	COPY, DELETE, PRINT, PROTECT, RENAME, TYPE, UNPROTECT	/DATE:[:date],/NEWFILES
/D	DELETE	—
	PRINT, TYPE	/DELETE
/E	COPY, DELETE, PRINT, PROTECT, RENAME, TYPE, UNPROTECT	/WAIT
/F	COPY, RENAME,	/PROTECTION
	PROTECT	—
/G	COPY	/IGNORE
/H	COPY	/VERIFY
/I[:date]	COPY, DELETE, PRINT, PROTECT, RENAME, TYPE, UNPROTECT	/SINCE[:date]
/J[:date]	COPY, DELETE, PRINT, PROTECT, RENAME, TYPE, UNPROTECT	/BEFORE[:date]
/K:n	PRINT, TYPE	/COPIES:n
/M:n	COPY, DELETE	/POSITION:n
/N	COPY, RENAME	/NOREPLACE
/0	COPY	/PREDELETE
/P	COPY, DELETE, PROTECT, UNPROTECT	/EXCLUDE

Table 20–2 (Cont.): DCL Equivalents of PIP Utility Operations

CSI Option	DCL Command(s)	DCL Option(s)
/Q	COPY, DELETE, PRINT, PROTECT, RENAME, TYPE, UNPROTECT	/QUERY
/R	RENAME	–
/S	COPY	/SLOWLY
/T[:date]	COPY, PROTECT, RENAME, UNPROTECT	/SETDATE[:date]
/U	COPY	/CONCATENATE
/V	COPY	/MULTIVOLUME
/W	COPY, DELETE, PRINT, PROTECT, RENAME, TYPE, UNPROTECT	/LOG
/X	COPY, DELETE, PRINT, PROTECT, RENAME, TYPE, UNPROTECT	/INFORMATION
/Y	COPY, DELETE, PRINT, PROTECT, RENAME, TYPE, UNPROTECT	/SYSTEM
/Z	COPY, RENAME, UNPROTECT	— /NOPROTECTION

Chapter 21
Queue Utility (QUEUE)

The Queue Utility (QUEUE) sends files to any valid RT–11 device. Although the Queue Utility is particularly useful for queuing files for printing, queuing is not restricted to a line printer or any other serial device.

The Components of the Queue Utility—the Queue Package

The Queue utility consists of the following three components.

Component	File	Function
QUEUE	SY:QUEUE.REL	Queues and sends the files you specify; runs as a foreground or system job.
QUEMAN	SY:QUEMAN.SAV	Processes command lines and file specifications you enter, and sends that information to QUEUE. QUEMAN runs as a background job and serves as the interface between you and the Queue utility.
QUFILE	SY:QUFILE.WRK	Contains the queue for the lineup of files waiting to be output to the device(s) you specify; this is also called Queue's work file.

The Queue utility runs only with a mapped monitor.

NOTE

To prevent QUEUE and another job from intermixing output on the same non-file-structured device, use the LOAD command to assign exclusive ownership of a device to QUEUE.

Features

- QUEUE appends a form-feed character <FF> to the end of each copy of a queued file, whether the output is to a disk, a serial line printer, or a parallel line printer.
- When QUEUE sends a job consisting of more than one input file to an RT–11 file-structured device, QUEUE now copies each input file to a separate output file with the same file name and type. The job name is printed in the JOBNAME field of the banner page.
- To save time, magtape input devices for QUEUE operations do not rewind between files.

- The PRINT command is affected when both QUEUE and SPOOL are running.
  - — KMON assigns precedence to SPOOL for any PRINT command, so take care if you run both QUEUE and SPOOL. PRINT options /PROMPT and /NAME are specific only to QUEUE. If both QUEUE and SPOOL are running, KMON treats those PRINT options as assigned to SPOOL and returns an invalid option error.
  - — When SPOOL or both SPOOL and QUEUE are running, the /FLAGPAGE:n option, when a value is specified for n, overrides the SET SP FLAG=n command. When no value is specified for n with the /FLAGPAGE:n option, the value for n is set by the SET SP FLAG=n command. The /NOFLAGPAGE option inhibits flag pages under all circumstances.
  - — When only the QUEUE package is running, the default number of banner pages printed when you use the /FLAGPAGE:n option is determined by the default number of banner pages set with the QUEMAN /P option. If the default set with the /P option is 0, the default for /FLAGPAGE:n is 1. If the QUEMAN /P option is not used, the default is /NOFLAGPAGE.
- If the QUEUE utility is running, the DCL command SHOW QUEUE requires RESORC.SAV and QUEMAN.SAV be on device SY.

Calling and Using the Queue Utility

To use the Queue utility, you must first run QUEUE from the system volume as either a foreground or system job. You can then run QUEMAN in the background when you are ready to output files.

Running QUEUE

- To run QUEUE as a foreground job, call QUEUE from the system volume issuing the following command:

  ```
  .FRUN QUEUE RET
  ```

- To run QUEUE as a system job, call QUEUE from the system volume issuing the following command:

  ```
  .SRUN QUEUE RET
  ```

To halt QUEUE, enter the /A option; see the description of the /A option for more information.

Running QUEMAN

To run QUEMAN from the system volume, issue the following command:

```
.R QUEMAN
```

The Command String Interpreter (CSI) displays an asterisk at the left margin of the terminal, indicating it is ready to accept input. Enter a command string according to this general syntax:

[dev:[jobname[/options]]=][filespec][/options][,filespec[/options]...]

where:

dev:	specifies any valid RT–11 device; the default output device is LP0.
jobname	specifies the output job name. This is the logical name for all the files specified in the command. If you send a job to a file-structured device, QUEUE uses this name as the file name of the job, and assigns a JOB file type. If you do not specify a job name, QUEMAN uses the file name of the first input file. The job name can have up to six characters.
filespec	specifies the input file. If you do not specify a file type, QUEMAN assumes a LST file type.
options	specifies one or more of the options listed in the QUEMAN Option Summary section.

If you use commas in place of file specifications, QUEMAN ignores all remaining file specifications on that command line. (Note, however, that if your command string consists of several lines, entering commas in place of a file specification does not affect file specifications on subsequent lines in the command string.)

QUEMAN Option Summary

Table 21–1 summarizes the options you can use in the QUEMAN command line. The sections that follow Table 21–1 provide detailed explanations and examples of each option. Note that some of the options are position-dependent—that is, their function depends on where you place them in the command line. Also, some of the options accept a date as an argument. The syntax for specifying the date is:

[:dd.][:mmm][:yy.]

where:

dd.	specifies the day (a decimal integer in the range 1–31).
mmm	specifies the first three characters of the name of the month.
yy.	specifies the year (a decimal integer in the range 73–99).

The default value for the date is the current system date. If you omit any of these values (dd, mmm, or yy), RT–11 uses the values from the current system date. For example, if you specify only the year ::90. and the current system date is May 4, l991, RT–11 uses the date 4:MAY:90. If the current date is not set, it is considered 0 (the same as for an undated file in a directory listing). The date values are position-dependent. If you omit the day (dd) or month (mmm), you must use a colon (:) in place of the value.

If you have selected timer support through the system generation process, but have not selected automatic end-of-month date advancement, make sure that you set the date at the beginning of each month with the DATE command. If you fail to set the date at the beginning of each month, the system prints –BAD– in the creation date column of each file created beyond the end-of-month. (Note that you can eliminate –BAD– by using the RENAME/SETDATE command after you set the date.)

Two Types of QUEMAN Options

QUEMAN options are of two types: action and qualifier. You should specify only one action option on a command line. If you specify more than one action option, only the last one specified is executed.

You can specify qualifier options in combination with each other and in combination with action options. However, if you specify an action option with a qualifier option, specify the action option first. Qualifier options should be placed after an action option on the command line. The reason is that QUEMAN executes the last specified option first; and when it executes an action option, QUEMAN ignores any further options specified on the command line.

Table 21–1: QUEMAN Option Summary

Option	Type	Function
/A	Action	Terminates QUEUE.
/C[:date]	Qualifier	Prints only those files with the specified date. If you use /C and do not specify a date, QUEMAN prints only those files with the current date.
/D	Qualifier	Deletes the input file(s) after printing. This option is position-dependent.
/H:n	Qualifier	Prints n banner pages for each specified input file, where n is a decimal number. This option is position-dependent.
/I[:date]	Qualifier	Prints only those files created on or after the specified date.
/J[:date]	Qualifier	Prints only those files created before the specified date.
/K:n	Qualifier	Prints n copies of each specified file, where n is a decimal number. This option is position-dependent.
/L	Action	Lists the contents of the queue.
/M	Action	Removes a job from the queue.
/N	Qualifier	Specifies no banner pages for the input file(s).
/P	Action	Sets two Queue Package default values: the number of banner pages, and whether you want QUFILE.WRK deleted when you terminate QUEUE.
/Q	Qualifier	Causes QUEMAN to request confirmation that a particular file should be included in the operation. QUEMAN prints the name of each file that can be included in the operation. You must respond Y to include a particular file.
/R	Action	Resumes sending the current job after it has been suspended, or restarts the current file in the job being sent.
/S	Action	Suspends output at the end of the current file.
/W	Qualifier	Displays on the terminal a log of the files involved in the operation.
/X	Qualifier	Allows QUEMAN to continue processing instead of halting when it cannot find a file you specified in the command line.
//	Qualifier	Continues command on the next line.

If QUEUE is sending a job that has multiple input files to an RT–11 file-structured volume, QUEUE copies each input file to a separate output file with the same file name and type as the input file. The jobname is used in the JOBNAME field of the banner page (if you request banner pages).

QUEUE protects files until it has placed them in the queue. Under RT–11, a file cannot be copied to a device that contains a protected file with the same file name. For this reason, QUEUE cannot be used to transfer a file to a device that contains a file with the same file name.

QUEMAN Option Descriptions

Terminating QUEUE (/A)

When you type /A in response to the CSI asterisk, QUEMAN terminates QUEUE. If you use /A while a job is printing, QUEUE halts output. If QUEUE is running as a foreground job, using /A has the same effect as typing CTRL/F and two CTRL/Cs. If QUEUE is running as a system job, using /A has the same effect as typing CTRL/X and then specifying QUEUE as the system job to which you want to direct input, followed by two CTRL/Cs.

The following example terminates QUEUE:

```
.R QUEMAN
*/A
```

If you type CTRL/C twice to terminate QUEUE, this may take a few seconds because QUEUE performs the following I/O rundown before terminating:

- Waits for all current I/O transfers to complete
- Removes protection from the input file if it was unprotected before QUEUE began copying it to the output device
- Closes the work file if you have chosen to save the work file

Date Option (/C[:date])

The /C[:date] option prints only those files with the specified date. If no date is specified, only those files with the current date are printed. Specify /C only once in the command line; it applies to all the file specifications in the entire command. The following command prints on LP0: all files named ITEM1 and ITEM2 that also have the date March 20, 1991:

```
*ITEM1/C:20.:MAR:91.,ITEM2
```

Deleting Input Files After Printing (/D)

Use the /D option to delete input files after QUEUE has sent them. This option is position-dependent. If you use it with the job name, /D applies to all the input files. If you use it with an input file specification, /D applies only to that input file.

Input files are protected from deletion while QUEUE is copying them to the output device. The following example deletes all input files after they have been sent:

```
*MYJOB/D=FILE1,FILE2,FILE3
```

The following example deletes FILE1 and FILE3 but retains FILE2 after QUEUE has sent them:

```
*MYJOB=FILE1/D,FILE2,FILE3/D
```

Input files are protected from deletion while QUEUE is copying them to the output device. This protects input files from accidental deletion.

Printing Banner Pages (/H:n)

Use the /H:n option to print banner pages for the input files you specify, where n is a decimal number selecting the number of banner pages. This option is position-dependent. If you use /H:n with the jobname, QUEUE prints n banner pages for each input file. If you use /H:n with an input file specification, QUEUE prints n banner pages for that file, and prints the default number of banner pages for the remaining input files. (Note that you set the default number of banner pages with the /P option. If the default number of banner pages set with the /P option is 0, n defaults to 1.)

The sample command line that follows prints four banner pages for each input file:

```
*LAUGHN/H:4=ROWAN.TXT,MARTIN.TXT
```

The following sample command prints four banner pages for MARTIN.TXT and the default number of banner pages for ROWAN.TXT:

```
*LAUGHN=ROWAN.TXT,MARTIN.TXT/H:4
```

Note that QUEUE never prints a banner for the job; it prints banners only for the input files.

NOTE

When SPOOL and QUEUE are running, QUEMAN option /H returns an invalid-option error message. This option conflicts with SPOOL (PIP) options, and KMON assigns SPOOL precedence over QUEUE.

Since Option (/I[:date])

The /I[:date] option prints only those files created on or after the specified date. If you specify no date QUEMAN uses the current system date. The following command prints only those .MAC files on device DK: created on or after April 21, 1991:

```
**.MAC/I:21.:APR:91.
```

Before Option (/J[:date])

The /J[:date] option copies only those files created before the specified date. If you specify no date QUEMAN uses the current system date. The following command prints only those .MAC files on device DK: created before April 21, 1991:

```
**.MAC/J:21.:APR:91.
```

Printing Multiple Copies (/K:n)

Use the /K:n option to specify the number of copies of the input files you specify, where n is a decimal number. The /K:n option is position-dependent. If you use /K:n with the job name, QUEUE prints n copies of each input file. If you use /K:n with an input file specification, QUEUE prints n copies of that particular file.

The next command line prints four copies of LAUREL.LST and four copies of HARDY.LST:

```
*JOB/K:4=LAUREL,HARDY
```

The following sample command line prints four copies of LAUREL.LST and the default number of copies of HARDY.LST:

```
*JOB=HARDY,LAUREL/K:4
```

Listing the Contents of the Queue (/L)

Use the /L option to get a listing of the contents of the queue. The listing gives the output device, job name, input files, job status, and number of copies for each job that is in the queue. The job STATUS column prints P if the job is currently being sent, S if the job being sent is suspended, or Q if the job is waiting to be sent. If you have a large queue and your console is a video terminal, you can use the keyboard CTRL/S and CTRL/Q commands to control the scrolling of the listing.

The sample command line that follows lists the queue:

```
*/L
DEVICE     JOB       STATUS   COPIES   FILES

LP0:       LAB2      P        1        PASS3 .LST
                              2        PASS4 .LST
                              2        PASS5 .LST
LP0:       HODG      Q        3        MESMAN.DOC
MT1:       JUDITH    Q        2        PART1 .DOC
                              2        PART2 .DOC
LP0:       JOYCE     Q        1        SSM   .DOC
                                       DOCPLN.DOC
```

Removing a Job from the Queue (/M)

Use the /M option to remove a job from the queue. When you use this option, specify the job name followed by /M and the equal sign (=). The following example removes the job LAB4 from the queue:

```
*LAB4/M=
```

When you use /M, you do not have to specify the input files, only the job name. You remove all the files associated with the job name.

No Banner Pages Option (/N)

Use the /N option to specify that you do not want QUEUE to print any banner pages for the input file(s). Use /N if you have previously set the default number of banner pages with the /P option. The /N option is position-dependent; that is, if you use it after the job name, it applies to each input file. Use /N after an input file to apply to only the particular file.

The following example uses /N to specify no banner pages for each file in the job, MYJOB2:

```
*MYJOB2/N=PASS1,PASS2,PASS3
```

The /N option has the same effect as /H:0.

NOTE

When SPOOL and QUEUE are running, QUEMAN option /N returns an invalid-option error message. This option conflicts with SPOOL (PIP) options, and KMON assigns SPOOL precedence over QUEUE.

Setting Queue Package Defaults (/P)

Use the /P option to set defaults for two values:

- Number of banner pages printed for each input file.

  You can override the default number of banner pages by using the /H option.

- Whether you want the work file, QUFILE.WRK, deleted when you halt QUEUE.

  Note that QUFILE.WRK contains the lineup of files, or queue, waiting to be sent to an output device.

When you type /P in response to the CSI asterisk, QUEMAN displays the following prompt at the terminal:

```
1) Number of banner pages ?
```

QUEUE uses the number you type as the default number of banner pages it prints for each file it sends to a device. If you type only a RETURN, QUEMAN assumes 0. This value remains in effect until the work file, QUFILE.WRK, is deleted (see below).

After you have responded to the previous prompt, QUEMAN prints the following prompt at the terminal:

```
2) Delete workfile ?
```

If you type N followed by a RETURN, or only a RETURN, QUEUE maintains the current QUFILE.WRK after you halt QUEUE. That is, if you start QUEUE later, QUFILE.WRK retains the queue it had prior to the halt. By maintaining QUFILE.WRK between the times QUEUE is halted, you have an automatic queue restart capability. This value remains in effect until you reset it.

If you type Y followed by a RETURN, QUEUE deletes QUFILE.WRK when you halt QUEUE. The next time you start QUEUE, it creates a new QUFILE.WRK. This value remains in effect only until the next time you start QUEUE.

Query Option (/Q)

Use the /Q option to list all files and to confirm individually which of these files should be printed. Typing Y or any string beginning with Y followed by a RETURN causes the named file to be printed. Typing anything else excludes the file. The following example prints files that reside on DY1:

```
*DY1:*.*/Q
DY1:FIX463.SAV    to LP: ?
DY1:GRAPH.BAK     to LP: ?Y
DY1:DMPX.MAC      to LP: ?
DY1:MATCH.BAS     to LP: ?
DY1:EXAMP.FOR     to LP: ?
DY1:GRAPH.FOR     to LP: ?Y
DY1:GLOBAL.MAC    to LP: ?Y
DY1:PROSEC.MAC    to LP: ?Y
DY1:KB.MAC        to LP: ?
DY1:EXAMP.MAC     to LP: ?
```

Suspending Output (/S)

Use the /S option to suspend output of a job being sent. When you type /S in response to the CSI asterisk, QUEUE suspends output only after it has completed output of the current file in the job. This option is useful if you want access to an output device while a large job is being sent to it.

To resume output, use the /R option.

Resuming/Restarting Output (/R)

Use the /R option either to resume output of a suspended job, or to restart output of the current file in the job from the beginning of the file. Note that a job resumes if you previously suspended it with /S, and a job restarts if you have not previously suspended it.

Resuming a job with multiple input files when the job is being sent to an RT–11, file-structured volume can be useful if the volume involved is too small to contain the entire job. You can suspend the job being sent (using the /S option), change volumes, and resume output of the remainder of the job on the new volume. QUEUE uses the same file name for both parts of the job.

Log Option (/W)

When you use the /W option, QUEMAN prints a list of all files printed or copied to a file. The /W option is useful if you do not want to take the time to use the query mode (the /Q option, but you do want a list of the files printed or copied by QUEMAN.

QUEMAN prints the log for an operation on the terminal under the command line. This example shows logging when files are queued to be printed on LP0:

```
*DY1:*.*/W
Files queued:
DY1:TEST.MAC     to LP:
DY1:FIX463.SAV   to LP:
DY1:GRAPH.BAK    to LP:
DY1:DMPX.MAC     to LP:
DY1:MATCH.BAS    to LP:
DY1:EXAMP.FOR    to LP:
DY1:GRAPH.FOR    to LP:
DY1:GLOBAL.MAC   to LP:
DY1:PROSEC.MAC   to LP:
DY1:EXAMP.MAC    to LP:
```

Information Option (/X)

The /X option causes QUEMAN to print an information message when QUEMAN fails to find all of the files you specify in a command line. If you do not use /X, QUEMAN prints a fatal error message when it is unable to find an input file, and returns control to the keyboard monitor after the rest of the operation completes. Use /X in indirect command files to ensure that processing will continue even if QUEMAN fails to find a file you specify.

In the following example, QUEMAN is unable to find the file FILE2.MAC. QUEMAN prints a message informing you that the file was not found and continues processing:

```
*LP:*.*=DL0:FILE1.MAC,FILE2.MAC,FILE3.MAC
?QUEMAN-I-File not found DL0:FILE2.MAC
```

Continuing a Command String (//)

Use the // option to continue a command string on subsequent lines. This option is useful if you want to output more files than you can specify on one line. When you want to include several lines in a CSI command string, type // at the end of the first line, and again at the end of the command string.

The following command string uses the // option:

```
*JOBNAM=LML1.MAC,LML2A.MAC//
*LML51.MAC,LML95.MAC
*LML4.MAC,LML56.MAC//
```

Putting Support for QUEUE in an Application Program

Usually you queue files that you want to copy to another device by using the monitor PRINT command. If the QUEUE program is running when you issue the PRINT command, the files you specify are queued automatically and the monitor dot prints on your terminal almost immediately.

Your application programs can also copy files to output devices through the QUEUE program. The method your program must use to do this depends on which monitor is currently running. If a monitor that includes the system job feature is running, your program must communicate with QUEUE through the message queue (MQ) handler by using .LOOKUP, .WRITW, and .READW programmed requests. Using the MQ handler is beneficial because it frees the monitor for other tasks, and takes advantage of the existing queued I/O system. Note that the MQ handler in a mapped system borrows kernel PAR2 for its own use if the conditional assembly parameter MQH$P2 = 1; see *RT–11 System Internals Manual* for more information on this topic.

If a monitor without the system job feature is running, your program must communicate with QUEUE through the .SDAT and .RCVD programmed requests.

To queue one or more files, follow these steps:

1. Set up a job block in your program for a logical group of files to be queued.
2. Set up a file block for each file to be queued.
3. Issue the .LOOKUP programmed request for the QUEUE program. (Omit this step if your system does not have the system job feature.)
4. Issue the .WRITW request (or the .SDATW request if your system does not have the system job feature) to send the QUEUE request and establish a pointer to the job and file blocks.
5. Issue the .READW request (or the .RCVDW request if your system does not have the system job feature) to receive acknowledgment from QUEUE.

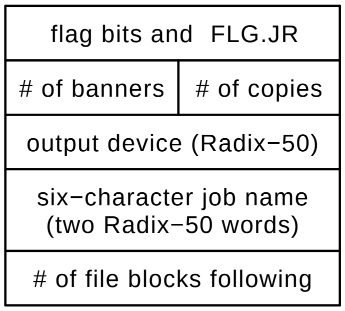
Once QUEUE acknowledges your request, your program is free to continue processing or to exit. Figure 21–5 shows a program that uses .LOOKUP, .READW, and .WRITW to queue one file, then exits.

Step 1: Setting Up the Job Block

Set up a job block in memory for a logical group of files. The job block defines the logical name by which you can later reference the entire group of files.

When you copy files to a file-structured device (such as the line printer, for example) all the files belonging to the job are concatenated and stored in a file called "jobname" with the file type .JOB. The handler for the device to which you send the job must be made resident in memory through the monitor LOAD command. Figure 21–1 shows the format of the job block.

Figure 21–1: Job Block



The flag word in each job block defines the action QUEUE should take on each file. Table 21–2 lists the definitions of the bits. Bits 4 through 15 are reserved for Digital.

The job block must have bit FLG.JR set. If FLG.CP is set, QUEUE sets the default number of copies to queue for this job from the low byte of the second word in the job block. If FLG.HD is set, QUEUE sets the number of banners to queue for this job from the high byte of the second word in the job block.

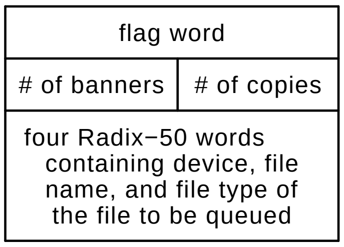
Table 21–2: Request Flag Bits

Bit	Name	Mask	Meaning
0	FLG.DE	1	Delete file after copying it.
1	FLG.CP	2	Make multiple copies (get number of copies from second word in block.)
2	FLG.HD	4	Create banner pages (get number of pages from second word in block).
3	FLG.JR	10	For initial request and job block.

Step 2: Setting Up the File Block

Immediately after the job block, your program must set up a file block for each file that is part of the job. Arrange the blocks contiguously in memory, with the job block first. Figure 21–2 shows the format of the file block.

Figure 21–2: File Block

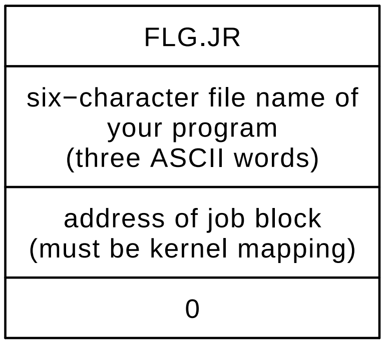

flag word
# of banners
# of copies
four Radix–50 words containing device, file name, and file type of the file to be queued

In each file block, you can specify the number of banner pages and the number of copies for the file by setting flag bits FLG.CP and FLG.HD, and putting values into the second word of the block. If you omit the flag bits, QUEUE ignores the second word of the file block and checks the flag bits of the job block instead. If they are set, QUEUE takes the values from the second word of the file block. Finally, if the flag bits are clear in both the file and the job blocks, QUEUE uses the system default of no banners and one copy of the file, or the current default parameters as set by the QUEMAN /P option.

Step 3: Setting Up the QUEUE Request Block

The last data structure you must establish is called the QUEUE request block. It need not be contiguous in memory with the job and file blocks. Figure 21–3 shows the format of the QUEUE request block. This block contains the information that QUEUE needs to begin processing the files. QUEUE requests can only be issued from a privileged job with kernel mapping. QUEUE request blocks must reside in low memory.

Figure 21–3: QUEUE Request Block


FLG.JR
six−character file name of your program (three ASCII words)
address of job block (must be kernel mapping)
0

Steps 4 and 5: Issuing Two Requests

- **Issuing the .LOOKUP Request**

  In the executable section of your program, you must issue a .LOOKUP programmed request to make the first contact with the QUEUE program and establish a communication channel. Issue the .LOOKUP for MQ:QUEUE, following the example provided in the *QUEUE Example Program* section. (Omit this step if your system does not have the system job feature.)

- **Issuing the Request to QUEUE**

  If the .LOOKUP is successful (or if you omitted it), you next issue the .WRITW programmed request (or the .SDATW request if your system does not have the system job feature) to send your request to QUEUE. The text you send to QUEUE is the QUEUE request block. See the example provided in *QUEUE Example Program* section.

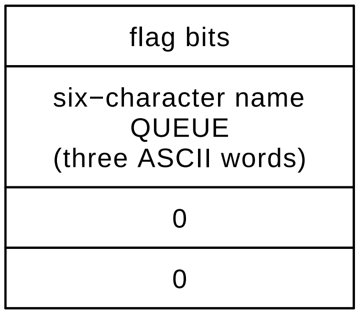
  If your request is valid, QUEUE inserts the request blocks into the queue, which is a workfile on device DK. The workfile is a first-in/first-out list; it can contain requests for different output devices. QUEUE does not maintain a separate workfile for each device.

Receiving Acknowledgment from QUEUE

When QUEUE acknowledges your request, your program can continue execution, or exit, as you desire. You obtain this acknowledgment by issuing the .READW programmed request (or the .RCVDW request if your system does not have the system job feature). QUEUE's response takes the form shown in Figure 21–4.

Your program must wait for this acknowledgment. QUEUE maintains only a limited number of extra queue elements. If QUEUE sends a message to your program that your program is not prepared to accept, a queue element is needlessly kept out of the list of available elements; this could block another job in your system.

Figure 21–4: Request Acknowledgment Block



If the acknowledgment is positive, the flag word contains 0. If the acknowledgment is negative, the sign bit of the flag word is set in addition to one of the low three bits. Table 21–3 shows the meanings of the acknowledgment flag bits.

Table 21–3: Acknowledgment Flag Bits

Bit	Name	Mask	Meaning
0	FLG.RA	0	Request accepted.
15,0	FLG.IR	100001	Illegal job request.
15,1	FLG.QF	100002	Insufficient room in workfile.
15,2	FLG.NQ	100004	QUEUE being aborted from the terminal.

QUEUE Example Program

Figure 21–5 contains a listing of an example program, MYPROG, that uses QUEUE in a system with the system-job feature to copy a data file to the line printer.

Figure 21–5: QUEUE Example Program

```
        .TITLE  MYPROG.MAC
        .ENABL  LC

; This example shows how an application program can
; send files through the queue system.

        .MCALL  .READW, .WRITW, .LOOKUP, .EXIT, .PRINT

        ;Flag bits for request

        FLG.DE= 1                       ;Delete file after printing
        FLG.CP= 2                       ;Multiple copies
        FLG.HD= 4                       ;Banner pages
        FLG.JR= 10                      ;Job request indicator

        .PSECT  QUETST

;Execution Section

START:  .LOOKUP #AREA,#16,#LKUP         ;.LOOKUP QUEUE
        BCC     1$                      ;Error?
        .PRINT  #LUPERR                 ;Yes, report it
        .EXIT                           ;and quit

1$:     .WRITW  #AREA,#16,#REQST,#6     ;Send initial
                                        ;request to QUEUE
        BCC     2$                      ;Error?
11$:    .PRINT  #REQERR                 ;Yes, report it
        .EXIT                           ;and quit

2$:     .READW  #AREA,#16,#REPLY,#6     ;Wait for ACK
                                        ;from QUEUE.  Word count
                                        ;of ACK in REPLY, text
                                        ;in REQST.
        BCS     11$                     ;Branch on error
        TST     REQST                   ;ACK okay?  (First word
                                        ;of ACK should be 0)
        BNE     MERR                    ;Branch if error
        .PRINT  #ACKMSG                 ;Print success message
        .EXIT                           ;End of test, request
                                        ;sent to line printer.

MERR:   .PRINT  #NAKMSG                 ;Print error message
        .EXIT                           ;and quit

        .PSECT  QUEDTA

;Block for .LOOKUP on QUEUE

LKUP:   .RAD50  /MQ /
        .ASCIZ  /QUEUE/
```

Figure 21–5 (continued on next page)

Figure 21–5 (Cont.): QUEUE Example Program

```
AREA:   .BLKW   5                   ;EMT area

;ACK from QUEUE goes here:
REPLY:  .WORD   0                   ;Word count from .READW
REQST:  .WORD   FLG.JR              ;Initial request
        .ASCII  /MYPROG/            ;Calling program
        .WORD   JOBBLK              ;Addr of job block
        .WORD   0                   ;End of initial request

;Block for job

JOBBLK: .WORD   <FLG.JR+FLG.HD+FLG.CP>  ;Flags for job,
                                    ;banners, and copies
        .BYTE   2,3                 ;2 copies, 3 banners
        .RAD50  /LP /               ;Send to printer
        .RAD50  /DATA  /            ;Logical job name
        .WORD   1                   ;One file follows:
FILBLK: .WORD   0                   ;No flags, use defaults
        .BYTE   0,0                 ;Default banners, copies
        .RAD50  /DK /               ;Filespec to be queued
        .RAD50  /TSTFIL/
        .RAD50  /DAT/

;Messages

LUPERR: .ASCIZ  /MYPROG-F-QUEUE not running/
REQERR: .ASCIZ  /MYPROG-F-Initial request error/
NAKMSG: .ASCIZ  /MYPROG-W-QUEUE acknowledgment negative/
ACKMSG: .ASCIZ  /MYPROG-I-QUEUE acknowledgment OK/
        .EVEN

        .END    START
```

DCL Equivalents of Queue Utility Operations

Table 21–4 lists the DCL PRINT and DELETE commands that are equivalent to Queue utility (QUEMAN) operations.

Table 21–4: DCL Equivalents of Queue Utility Operations

CSI Option	DCL Command	DCL Option
A	no equivalent	
C[:date]	PRINT	/DATE[:date] /NEWFILES
/D	PRINT	/DELETE
/H:n	PRINT	/FLAGPAGE:n
/I[:date]	PRINT	/SINCE[:date]
/J[:date]	PRINT	/BEFORE[:date]
/K:n	PRINT	/COPIES:n
/L	SHOW	/QUEUE
/M	DELETE	/ENTRY
/N	PRINT	/NOFLAGPAGE
/P	no equivalent	
/Q	PRINT	/QUERY
/R	no equivalent	
/S	no equivalent	
/W	PRINT	/LOG
/X	PRINT	/INFORMATION
//	PRINT	/PROMPT

Note that the SHOW QUEUE command is performed by the RESORC /Q option, rather than by the QUEMAN /L option. However, the QUEMAN /L option is still valid for compatibility and is equivalent to the SHOW QUEUE command.

Chapter 22

System-Resource Display Utility (RESORC)

The System-Resource Display Utility (RESORC) examines the currently running RT–11 system and displays information about the status of the monitor and the system configuration. You can use RESORC to display on your terminal the following data about RT–11:

- Hardware configuration
- Monitor version
- Total amount of memory on the system and organization of physical memory
- Special features in effect
- Device names and logical device-name assignments
- Whether a device is assigned as a default device
- Terminal characteristics for terminals currently active on a multi-terminal system
- Logical-disk subsetting
- Device handler status
- If you are running the Error Logger, QUEUE, or SPOOL, RESORC can provide information on:
  - Errors
  - The update status of files waiting to be sent to an output device
  - User-defined commands

Calling and Terminating RESORC

To call RESORC from the system device, respond to the keyboard monitor dot (.) by typing:

```
.R RESORC  RET
```

The Command String Interpreter (CSI) displays an asterisk (*) at the left margin of the terminal and waits for your input. At this point, enter the RESORC option or options required to obtain the information you need.

If you enter only a RETURN in response to the asterisk, RESORC displays its name and current version number. To abort RESORC while it is executing, enter CTRL/C twice. Type CTRL/C once to return control to the monitor when RESORC is waiting for input (that is, when an asterisk has displayed on the terminal).

RESORC Option Summary

Table 22–1 summarizes the RESORC options by listing what each one displays.

Table 22–1: RESORC Options

Option	Display
/A	The result of a combination of all RESORC options (except /Z)
/C	The device from which you bootstrapped the system and the monitor SET options in effect
[dd:]/D	A list of the device handlers, their status, and their vectors; when [dd:] is included, lists the status of only that device
/H	The hardware configuration, including the system's total amount of memory (in bytes)
/J	Information about the currently running and loaded jobs
/L	Device assignments
/M	The monitor type, version number, and update level
/O	The system generation special features in effect
/Q	The contents of the queue for QUEUE or SPOOL or both
/S	Information about logical-disk subsetting
/T	The status and options in effect for currently active terminals on multi-terminal systems
/V	The release and version number of any module in the RT–11 distribution kit.
/X	The organization of physical memory
/Z	The result of a combination of /M, /C, /H, /J, and /O options

By specifying one or more of the options /C, /D, /H, /J, /L, /M, /O, /S, /T, and /X, you choose the information that RESORC lists on the terminal. If you use two or more options, you can enter them in any order, although RESORC lists the information in the order /M, /C, /H, /O, /D, /L, /J, /T, /X, /S.

RESORC offers two options that are equivalent to combinations of options:

- The /Z option has the same effect as a combination of the /M, /C, /H, /J, and /O options.
- The /A option has the same effect as a combination of all the options except /Q and /Z.

RESORC Option Descriptions

All Option (/A)

The /A option has the same effect as a combination of all the other RESORC options (except /Z). When you enter /A, all RESORC information is displayed on the terminal.

Software-Configuration Option (/C)

The /C option displays status information for

- The device from which you bootstrapped the system
- Whether a foreground job is loaded (if applicable)
- The monitor SET options

  SET KMON
  SET EXIT
  SET EDIT
  SET SL
  SET CLI
- Command-file nesting depth
- Extended device-unit support
- Global .SCCA flag support (enabled or disabled)

The following example uses the /C option:

```
*/C

Booted from DU0:RT11XM

USR is set NOSWAP
EXIT is set SWAP
KMON is set NOIND
TT is set NOQUIET
ERROR is set ERROR
SL is set ON
EDIT is set KEX
KMON nesting depth is 3
CLI is set DCL, CLI, UCL, no UCF
```

Device-Handler Status Option ([dd:]/D)

The /D option displays:

- The RT–11 device handlers and their status
- CSR addresses and vectors
- Installability information for any device handler that, because of the handler characteristics or your system configuration, cannot be installed in some manner on your system.

The three possible displays are:

Display	Meaning
`Not BSTRAP installable`	`Handler must be installed by INSTALL command`
`Not KMON installable`	`Handler must be installed at system boot`
`Not installable`	`Handler cannot be installed on this system`

You can obtain this information for a specific device by including the optional argument *dd*. The *dd* variable specifies the two-letter permanent device mnemonic.

- The port display indicates an invalid port by displaying an asterisk (*) before the port number. For example, port = *n, where n = 0,1,2,3, indicates that port n was not installed. If RESORC option DU:/D displays port = n, where n = 0,1,2,3, then port n is installed.
- The default device, if present, by displaying an asterisk (*) next to the device handler or logical disk unit. (A default device is created by using the ASSIGN dev * command.)

DU:/D displays additional DU status information. The port display indicates an invalid port by displaying an asterisk (*) before the port number. For example, port = *n, where n = 0,1,2,3, indicates that port n was not installed. If DU:/D displays port = n, where n = 0,1,2,3, then port n is installed.

MU:/D displays MU status information for the PORT and UNIT of each installed TMSCP controller in the same manner that the DU argument displays MSCP status information. (Partitioning is invalid with magtape devices.)

The following messages in a /D display relate to a device's installation status.

Message	Meaning
Installed	Loaded into memory
nnnnnn	Load address of handler
Resident	Permanently located in memory
Not installed	Not loaded into memory
Not installed	Not loaded because the handler special features do not match those of the monitor
Not BSTRAP installable	Handler must be installed by the INSTALL command
Not KMON installable	Handler must be installed at the system boot
Not installable	Handler cannot be installed on this system

Hardware-Configuration Option (/H)

The /H option lists the processor type, the total amount of memory (in bytes) that the system contains, and all the special hardware features in your system configuration.

Any special hardware is chosen from the following list. (The /H option displays the features in the order they occur in the list.)

FP11 Hardware Floating Point Unit
Commercial Instruction Set (CIS)
Floating Point Microcode
Extended Instruction Set (EIS)
Floating Point Instruction Set (FIS)
Memory Management Unit
Parity Memory
ECC Memory
Cache Memory
PMI Memory
VT11 or VS60 Graphics Hardware
Extended Arithmetic Element (EAE)

The next item that appears in the /H listing is the clock frequency (50 or 60 cycles), and the last is the KW11–P programmable clock (if your system has one and you are not using it as the system clock).

The following example shows the /H option:

```
*/H

PDP 11/23 PLUS Processor
1024KB of memory
Floating Point Microcode
Extended Instruction Set (EIS)
Memory Management Unit
60 Cycle System Clock
```

Loaded-Jobs Option (/J)

The /J option displays information about the currently loaded jobs. For each job, RESORC displays:

- Job name and number (if you have not enabled system-job support on your monitor, the foreground job name appears as FORE, and its priority is 1)
- Console the job owns (with a non-multiterminal monitor, this space is blank)
- Priority level of the job
- Job’s running state (running, suspended, or done but not unloaded)
- Low and high memory limits of the job
- Start address of the job’s impure area

Device-Assignments Option (/L)

The /L option displays your system's device assignments. The devices RESORC lists are those in the system tables. The listing also includes additional information about particular devices. The informational messages and their meanings follow.

Message	Meaning
(RESORC) *or* =RESORC	The device or unit is assigned to the background job RESORC (for multi-terminal unmapped or mapped monitors only).
(F) *or* =F	The device or unit is assigned to the foreground job (for only multi-terminal unmapped monitors, mapped monitors, and monitors without system-job support).
(jobname) *or* =jobname	The device or unit is assigned to the system or foreground job (for FB and mapped monitors that have system-job support), where jobname represents the name of the system or foreground job.
(Loaded)	The handler for the device has been loaded into memory with the LOAD command.
(Resident)	The handler for the device is included in the resident monitor and is not unloadable.
=logical-device-name(1), logical-device-name(2), ...logical-device-name(n)	The device or unit has been assigned the indicated logical device names with the ASSIGN command.
xx free slots	The last line tells the number of unassigned, or free, device slots.

The following is an example display with the /L option:

```
*/L  RET
TT  (Resident)
DU  (Resident)
  DU0   = SY, DK, OBJ, SRC, BIN
  DU1   = LST, MAP
MQ  (Resident)
DL  (Loaded)
DM
DX  (Loaded)
  DX0:  (MYPROG)
LP: (Loaded=QUEUE)
MT
5 free slots
```

The listing shows first that TT, MQ, and RK are resident in memory. The other device handlers known to the system are DL, DM, DX, LP, and MT. There are five free slots in the table. DU0 has the logical names SY, DK, OBJ, SRC, and BIN. DU1 has the logical names LST and MAP. The DX handler is loaded and device DX0 belongs to the foreground job, MYPROG. The LP handler is loaded and belongs to the system job, QUEUE.

Current-Monitor Option (/M)

The /M option displays the type, version number, and update level of the currently running monitor. The following table identifies each RT–11 5.6 monitor:

Monitor	**Type**
SB	single-job, unmapped
FB	multi-job, unmapped
XB	single-job, single mapped
XM	multi-job, single mapped
ZB	single-job, fully mapped
ZM	multi-job, fully mapped

The following example uses the /M option:

```
*/M

RT-11XM  V05.06
```

Special-Features Option (/O)

The /O option lists on the terminal the special features chosen during a system generation. Whatever features are in effect are displayed in the same order as the following list of possible special features.

Option	Function
Device I/O timeout support	Permits device handlers to do the equivalent of a mark time without doing a .SYNCH request; DECnet applications require this support.
Error-logging support	Keeps a statistical record of all I/O operations on devices that are supported by this feature; detects and stores data on any errors that occur during I/O operations.
Multi-terminal support	Permits you to use as many as 16 terminals.
Memory-parity support	Causes RT–11 to display an error message when a memory-parity error occurs.
SB timer support	Configures the SB monitor to include mark-time and cancel mark-time programmed requests and to support the .FORK process.
System-job support	Allows you to run up to six jobs in the foreground in addition to the foreground and background jobs.
Global .SCCA support	Reports whether global .SCCA support was chosen during system generation.
Floating-point unit support	Reports whether FPU support was chosen during system generation.
Multi-terminal handler hooks support	Reports whether the monitor was SYSGENED for multi-terminal handler hooks.
Extended unit support	Reports whether the monitor was SYSGENED for extended unit support.
Unibus mapping	Reports whether it is available and enabled or disabled.

The following example shows the /O option:

```
*/O

Device I/O time-out support
Error logging support
System job support
FPU support
```

If there are no special features in effect, RESORC displays *NO SYSGEN options enabled*.

Show-Queue Option (/Q)

The /Q option lists the contents of the queue for QUEUE, SPOOL, or both, if both are running. If there are no files in a queue, RESORC displays:

```
?RESORC-I-No queues active
```

SPOOL Status Report

The SPOOL status report shows whether each SPOOL device unit and associated output device is active or inactive, the number of blocks spooled for output, and the number of free blocks in SPOOL's work file.

The following example display shows two output devices, LS0 and LP0, attached to SPOOL, with LS0 printing and LP0 idle:

```
 Unit  Device  Status
 SP0:  LS0:    ACTIVE,      56 blocks spooled
 SP1:  LP0:    IDLE

   944 Free blocks in workfile
```

QUEUE Status Report

The QUEUE status report shows the output device, job name, input files, job status, and number of copies for each job that is queued. The next example command lists the current contents of the queue for QUEUE:

```
*/Q  RET

DEVICE  JOB   STATUS  COPIES  FILES

LP0:    LAB2   P        1      PASS3 .LST
                        2      PASS4 .LST
                        2      PASS5 .LST
LP0:    HODG   Q        3      MESMAN.DOC
MT1:    SZYM   Q        1      REFMAN.TXT
LP0:    JOYCE  Q        1      SSM   .DOC
                        1      DOCPLN.DOC
```

The job status column contains a P if the job is currently being output, an S if the job being output is suspended, or a Q if the job is waiting to be output. If you have a large lineup of files, and your console is a video terminal, you can use the CTRL/S and CTRL/Q commands to control the scrolling of the listing.

UNIBUS Mapping-Registers Option (/R)

The /R option displays the status of UNIBUS mapping registers (UMRs) if the UB pseudohandler is loaded, or information about why UB is not loaded. UNIBUS mapping register support is described in the *RT–11 System Internals Manual*.

The following is an example display.

```
UMR allocation
---------------------------------------------------
00 ...... 1  10 ...... 2  20 ...... 9  30 ..MS.. P
01 ...... 1  11 ...... 2  21 ...... 9  31 ...... P
02 ...... 1  12 ...... 2  22 ...... 9  32 ...... P
03 ...... 1  13 ...... 4  23 ...... 9  33 ...... P
04 ...... 1  14 ...... 4  24 ...... 9  34 ...... P
05 ...... 2  15 ...... 4  25 ...... 9  35 ..DU.. P
06 ...... 2  16 ...... 4  26 ...... 9  36 ..DU.. P
07 ...... 2  17 ...... 9  27 ...... 9  37 IOPAGE P

2. UMRs in use
2. UMRs permanently assigned
0. UMRs dynamically assigned

0. requests waiting for UMR allocation

RESORC = NOSERIAL
```

Disk-Subsetting Option (/S)

The /S option displays information about the subsetting of physical disks into logical disks. When you use the /S option, RESORC displays the following information:

- Assigned logical-disk units
- An asterisk next to any logical-disk unit assigned as a default device. (A default device is created by using the ASSIGN dev * command.)
- The file name to which each logical-disk unit is assigned
- The size of each logical disk in decimal blocks
- Any logical name assigned to a logical disk

For example, if you mount a 1000-block file DU4:WRK.DSK on logical device unit LD0 and assign it the name WRK, SHOW SUBSET displays:

```
LD0 is DU4:WRK.DSK[1000.] = WRK
```

The following sample command line displays the logical disks into which the physical disks DU and DL1 are divided:

```
.SHOW SUBSET RET
LD0 is DU:DISK.LST[4000.]
LD2 is DL1:DISK.SRC[1200.]
LD1 is DL1:WORK.DSK[600.]
```

An asterisk (*) following the file information indicates that although the logical-disk assignment exists, the file does not exist on the volume that is currently mounted in the drive unit. A number sign (#) indicates that the device handler is not loaded. These symbols are especially useful in determining the status of logical-disk assignments after you use the SET LD CLEAN command.

If LD.SYS is not installed, RT–11 displays the message *LD handler unavailable*. If no logical-disk units have been mounted (by using the MOUNT command), RT–11 displays *No LD units mounted*.

Terminal-Status Option (/T)

The /T option displays the status of and special features in effect for currently active terminals on multiterminal systems. If the monitor does not include multiterminal support, the following message displays:

```
No multi-terminal support
```

Since multiterminal support is not part of the distributed RT–11 monitors, such support is present on your system only if you have included it during system generation; that is, multiterminal support is a special feature.

If the monitor includes multiterminal support, /T displays a table of the existing terminals and lists the following information:

`Unit number:`	`0-16`
`Owner:`	`Background, foreground, system job, or none`

`Type:`	`Local`
	`Remote (dial-up)`
	`Console`
	`S-Console (shared by background and foreground or system job)`
	`Is attached to another job (the foreground)`

`Interface type:`	`DL, DZ, or DH`
`Width:`	`Width in characters, up to 255`

`SET options in effect:`	`TAB`
	`CRLF`
	`FORM`
	`SCOPE`

`Line speed:`	`Baud rate if DZ or DH; not applicable (N/A) if DL`

The following example shows the terminal status of an RT–11 system:

```
*/T RET

Unit Owner     Type      WIDTH TAB CRLF FORM SCOPE SPEED
--------------------------------------------------------
  0         Console   DL    80  No  No   No    Yes   N/A
  1         Local     DL    80  No  No   No    Yes   N/A
  9   XL*   Local     DH    80  No  No   No    Yes  9600
 10         Local     DH    80  No  No   No    Yes  9600
 11         Local     DH    80  No  No   No    Yes  9600
 12         Remote    DH    80  No  No   No    Yes  9600
    * Multi-terminal handler hooks in use
```

Note that in this table, the unit number refers to the terminal; RT–11 multiterminal support permits you to use as many as 17 terminals.

The Unit column lists the terminal unit number, and the Owner column lists the name of the job (foreground, system, background, or none) to which the terminal is assigned. If the running monitor does not have system job support, RESORC displays FORE and RESORC, where applicable.

The Type column of this table shows the type of terminal—local, remote, console, or S-console (a console shared between background, system, and foreground jobs)—and the type of hardware interface the terminal uses—DL, DZ, or DH.

The WIDTH column indicates the width in characters (up to 255) of the terminal listing or display text.

The next four columns indicate which SET options are in effect on the terminal. If you have set TAB, the terminal can execute hardware tabs. If you have set CRLF, the terminal issues a carriage return and line feed whenever you attempt to type past the right margin. The terminal is capable of executing hardware form feeds if you have set FORM and, on graphics terminals, capable of echoing RUBOUT characters as backspace-space-backspace if you have set SCOPE.

The last column, SPEED, lists the terminal's baud rate (if interface is DZ or DH). An N/A under the SPEED column indicates that the baud rate is not alterable (DL interface, with the exceptions of DLV11–E).

Version-Numbers Option (/V)

The /V option displays the release number of any system utility or handler in the RT–11 distribution kit and the version numbers for all modules in the utility or handler.

Use the /V option to supply the release and version numbers for any RT–11 utilities or handlers quoted in a Software Performance Report (SPR) submission.

In the following example, the /V option displays the release and version numbers for the LS handler:

```
*LS.SYS/V
Release = V05, Version(s) = 27
```

Physical-Memory Layout Option (/X)

The /X option lists the organization of physical memory. The memory listing shows the location of each low-memory component and, under a mapped monitor, each extended-memory region as well.

The memory listing displays such information as:

- Where jobs are loaded
- Where device handlers are loaded
- Where in memory KMON and the USR reside
- Number of words of memory each job and device occupy

- Type of region in the extended memory map
- Cache-bypass status for a global region, using the symbol BYP rather than the symbol GBL.

If you are running under a mapped monitor, the SHOW MEMORY listing is divided into two sections, the first for extended memory and the second for kernel memory. Memory addresses are displayed in octal.

The following example displays the organization of physical memory when running under the SB monitor:

```
*/X RET

Address   Module    Words
-------   ------    -----
160000    IOPAGE    4096.
157400    RK         120.
127274    RMON      6170.
126112    DY         313.
001000    ..BG..   21797.
```

The next example shows the organization of physical memory when running under the XM monitor:

```
*/X RET

------- Extended Memory --------
Address   Module    Words   Type
-------   ------    -----   ----
17760000  IOPAGE     4096.   PRM    HDW    BYP
02000000  MEMTOP
01000000  VM       131072.   SHR
00213700  ......    95264.
00201300  SL         2688.   PVT
00173200  MU         1568.   PVT
00164700  SP         1632.   SHR
00160000  DU         1248.   PVT
00000000  KERNEL    28672.   PRM    HDW

------ Low Memory -------
Address   Module    Words
-------   ------    -----
157340    DU          144.
117760    RMON       8056.
105716    USR        2577.
001000    ..BG..    17639.
```

The extended-memory display includes a symbol for the type of region listed. The table *Extended-Memory Region Types* shows the six types that can be listed. If the region has characteristics of more than one type, RESORC uses the following order of precedence:

- Private (PVT)
- Permanent (PRM)
- Automatic global elimination (AGE)

- Hardware (HDW)
- Shared (SHR)
- Local (LCL)

For example, a shared region that also has automatic global elimination enabled is displayed as type AGE. AGE takes precedence over SHR.

Extended-Memory Region Types

Symbol	Meaning	Description
AGE	Automatic global elimination	Created with the RS.AGE status argument set. Any program can attach to and access the region.
HDW	Hardware	Considered part of the hardware configuration. RT–11 runs only if this region is present.
LCL	Local	Created by a program for the exclusive use of that program. When the program exits or issues an .ELRG request, the region is eliminated.
PRM	Permanent	Permanently installed when RT–11 is bootstrapped. RT–11 runs only if this region is present.
PVT	Private	Created by a program that has not detached from the region. The creating program has exclusive use of the region. It cannot be attached to or accessed by another program.
SHR	Shared	Created by a program that has detached from it and possibly reattached to it. It can be attached to and accessed by any other program. The region remains after the creating program has exited. It can be eliminated by the REMOVE keyboard command or by specifying the RS.EGR status argument when a program issues the .ELRG request.

Summary Option (/Z)

The /Z option has the same effect as a combination of the /M, /C, /H, /J, and /O options. The /Z option displays the monitor version number and update level, the monitor SET options in effect, the hardware configuration, the total amount of memory on the system, and the special features in effect (if any). The listing varies, of course, depending on which monitor and which hardware system you are using.

Description of a CONFIGURATION Listing

- **Version number and update level**

  First, the listing always shows the version number and update level of the currently running monitor.

- **Monitor information**

  Next, the listing shows the following information about the monitor.

  — The first line indicates the device from which the system was bootstrapped

  — The next line indicates whether or not 22-bit addressing is on if you are running a mapped monitor.

  — Then the listing shows:

  Extended device-unit support
  SET CLOCK and SET FORTRA conditionals
  User service routine (USR) status: SWAP or NOSWAP
  EXIT status: SWAP or NOSWAP
  Active command file processor: KMON or IND
  SET RUN status
  SET MODE status
  Terminal status: QUIET or NOQUIET
  ERROR severity level
  SL status: ON, OFF, KMON
  Default editor for the EDIT command
  File nesting level (a decimal number)
  Status of .SCCA support and the .SCCA flag when enabled

- **System hardware configuration**

  Next, the listing shows the system hardware configuration. This includes:

  — The processor type, such as a MicroPDP–11 or a PDP 11/84

  — The total amount of memory your system contains; for example:

```
1022KB of memory
```

  — A separate line for each of the following items that is present on your system:

```
Extended arithmetic element (EAE)
FP11 Hardware Floating Point Unit
Commercial Instruction Set (CIS)
Extended Instruction Set (EIS)
Floating Instruction Set (FIS)
Memory Management Unit
Parity Memory
ECC Memory
Cache Memory
60 Hertz System Clock
```

  — Another line displays any graphics hardware (VT11 or VS60) you might have

  — The clock frequency (50 or 60 Hertz) displays next

  — The KW11–P programmable clock, if there is one on your system

— Finally, the listing either shows that there are no special features in effect, or it lists the features such as:

```
Device I/O time-out support
Error logging support
Floating Point Microcode
FPU support
Multi-terminal support
Memory parity support
SB timer support
System job support
Global .SCCA support
Unibus mappy support
multi-terminal handler support
extended unit support
```

An Example of the SHOW CONFIGURATION Display

The following example was created on a PDP 11/23 PLUS processor:

```
.SHOW CONFIGURATION

RT-11XM  V05.6
Booted from DU0:RT11XM
22 bit addressing is on

USR     is set NOSWAP
EXIT    is set SWAP
KMON    is set NOIND
TT      is set NOQUIET
ERROR   is set ERROR
SL      is set OFF
EDIT    is set KEX
FORTRAN is set FORTRA
KMON nesting depth is 3

CLI is set DCL, CCL, UCL, NO UCF

PDP 11/23 PLUS Processor
512KB of memory
Extended Instruction Set (EIS)
Memory Management Unit
Parity Memory
60 Hertz System Clock

Device I/O time-out support
System job support
FPU support
```

DCL Equivalents of RESORC Utility Operations

Table 22–2 lists the DCL SHOW command options that are equivalent to RESORC utility operations.

Table 22–2: DCL Equivalents of RESORC Utility Operations

CSI Option	DCL Command/Option
/A	ALL
/C	no DCL equivalent
[dev:]/D	DEVICES[dd:]
/H	no DCL equivalent
/J	JOBS
/L	SHOW (with no option)
/M	no DCL equivalent
/O	no DCL equivalent
/Q	QUEUE*
/R	UMR
/S	SUBSET
/T	TERMINALS
/V	no DCL equivalent
/X	MEMORY
/Z	CONFIGURATION

* The SHOW QUEUE command is performed by the RESORC /Q option, rather than by the QUEMAN /L option. However, the QUEMAN /L option is still valid for compatibility.

Chapter 23

RT Monitor Utility (RTMON)

The RT Monitor Utility (RTMON) is an unsupported utility that runs as a foreground job and provides a real-time display of system activity. It requires a VT100, VT200, or PC300 series terminal or system. RTMON runs only under monitors that include system-job support.

To use RTMON, type FRUN RTMON in response to the monitor prompt. RTMON requires no further commands but will respond to some control characters, such as CTRL/W to refresh the screen and CTRL/Z to clear the screen and suspend RTMON.

For best results, use a separate terminal for the RTMON display. This is possible only under monitors that include multiterminal support.

Chapter 24

Save-Image Patch Utility (SIPP)

The Save-Image Patch Utility (SIPP) enables you to examine code or to make code modifications to any RT–11 binary-output file that exists on a random-access storage volume.

Uses

- You use SIPP primarily for maintaining save-image files. Although SIPP is designed for maintaining programs that have been created with the RT–11 Version 4 or later linker, you can use SIPP for pre-Version 4 programs that are not overlaid.
- SIPP is useful for examining locations within a file. If you do not modify any locations within a file, SIPP makes no changes.

Features

- You can use SIPP to examine or patch (modify individual locations) in programs (overlaid or nonoverlaid) that were linked under RT–11 (Version 4.0 or later).
- SIPP allows you to patch nonoverlaid programs, programs with only low-memory overlays, programs with only extended-memory overlays, and programs with both low-memory and extended-memory overlays.

  Note the following, however, for dealing with overlaid programs:

  — When a program you specify for a SIPP session is overlaid, SIPP reads in 259_{10} words of that program at a time; that is 256_{10} words of a single block, plus the first three words of the next block. If either block is a bad block, SIPP returns the error message, *?SIPP-F-Input error <dev:filenam.typ>*. If you receive that error message, issue the command DIRECTORY/BAD for that device to find the bad block. SIPP reads in one block at a time when the program is overlaid and you use the /A option, or if the program is not overlaid.

  — If you use SIPP to patch or examine an overlaid program without using the /A option and SIPP reads to the end of a disk, SIPP returns the error message, *?SIPP-F-Input error <dev:filenam.typ>*, and does not read the last block on the disk.

- You can run SIPP from a command file, a BATCH stream, or from a terminal.
- When you run SIPP, you have the option of installing your code modifications when you close the file, or you can create a command file that contains both the code modifications and the instructions necessary for SIPP to install them. If

you create a command file, you can run this file as a command file whenever you wish.

- When SIPP patches a file, the creation date of the patched file is changed to the current system date.
- Because SIPP does not install code modifications until you have finished making them, SIPP's checksum is not affected by a CTRL/U or DELETE. This feature also makes the code modification, or patching, procedure easier for you.

NOTE

Digital does not recommend that you modify the following data within a save-image file: locations 50, 64, and 66; the Job Status Word; the overlay handler; the overlay tables; and the window definition blocks. SIPP uses these locations for internal calculations and will automatically update them as necessary. Note, however, that if you use the /A option, SIPP does not modify any of these locations.

Calling and Terminating SIPP

To call SIPP, respond to the dot (.) displayed by the keyboard monitor by typing:

```
.R SIPP  RET
```

The Command String Interpreter (CSI) displays an asterisk (*) at the left margin of the terminal and waits for a command string. If you only press RETURN in response to the asterisk, SIPP displays its current version number. If you press CTRL/C in response to the asterisk, control returns to the monitor. If you press CTRL/C in response to any of SIPP's prompts, SIPP displays the following confirmation message:

```
?SIPP - Are you sure?
```

If you type Y or any string beginning with Y followed by a RETURN, SIPP aborts the patching procedure, and returns control to the monitor, without making any changes to your file. Any other response returns control to the procedure that was interrupted. You must press CTRL/C twice at any other time, including while running from a command file, to get the *?SIPP—Are you sure?* message.

Command-Line Syntax

Enter a command string according to this general syntax:

[command-file=]input-file[/option...]

where:

command-file	specifies the command file that you want SIPP to create. You can run this file as a command file. The default file type is COM. If you do not specify a command file, SIPP does not create one.
input-file	specifies the file or volume you want to modify. If you do not specify a file type, SIPP assumes SAV. If you enter only a device specification, SIPP opens the first block of that volume and assumes the /A option.
/option	is one of the options listed in Table 24–1.

SIPP Option Summary

Table 24–1 summarizes the options that you can use with SIPP.

Table 24–1: SIPP Options

Option	Function
/A	Prevents SIPP from automatically modifying location 50, the window definition blocks, the overlay table, or the overlay handler. Use /A when you are patching anything other than save-image files. When you use the option, SIPP modifies only those locations that you specify. **Note:** Whenever you use SIPP to perform a customization on a distributed or generated RT–11 5.6 or later monitor, always use the /A option; this is needed because of the way the monitor is linked.
/C	Requires you to enter a checksum after you finish code modifications. If you make no modifications, SIPP ignores /C. The command file will automatically contain /C. You cannot use /C and /D together. See the section *Verifying Your Work with Checksum* for more details on the checksum.
/D	Use if you do not know the checksum for a particular patch and you want SIPP to create one. SIPP displays the checksum for the patch after you have finished entering all the code modifications. If you make no modifications, SIPP ignores the /D option. You cannot use /C and /D together.
/L	When you use /L, SIPP does not modify the input file after the patching session. This option is useful if you wish only to create a command file and preserve the input file.

No DCL Equivalents of SIPP Utility Operations

SIPP is not accessible through DCL commands.

SIPP Dialogue

After you have entered the initial command string to SIPP, SIPP displays a series of prompts at the terminal. The responses you give to these prompts guide SIPP to the location in the input file or volume where you want to begin code modifications.

Prompts

- If the input file is overlaid, the first prompt SIPP displays at the terminal is:

  ```
  Segment?
  ```

  Respond to this prompt by typing the number of the overlay segment that contains the locations you want to modify. (SIPP does not display this prompt if the file you are modifying is not overlaid, if you are using the /A option, or if you are modifying a volume.) You can find the segment number in the program's load map. Press RETURN or type 0 (zero) and RETURN, if you want to modify the program's root segment.

- SIPP prompts you for the base address within the program or overlay segment where you want to begin code modifications or examination. SIPP displays the following prompt for both overlaid and non-overlaid files. (Note that the following prompt is the second prompt for overlaid files, and the first prompt for anything else.)

  ```
  Base?
  ```

  If the file you are modifying is overlaid, respond to the preceding prompt by entering the base address specified on the load map for the segment you want to modify. If the file is not overlaid, enter the load address of the program section you wish to modify or examine.

- After you have entered the base address, SIPP prompts you for the offset as follows:

  ```
  Offset?
  ```

  Respond to the offset prompt by typing the offset from the current base where you want to begin modifying or examining your program.

  If the offset you specify is an even number, SIPP opens the corresponding location as a word. If the offset is odd, SIPP opens the location as a byte. The SIPP Command Descriptions section describes how you can alternate between words and bytes as you proceed to modify or examine the file.

- After you have responded to *Offset?*, SIPP displays the following header:

  ```
  Segment      Base      Offset      Old     New?
  ```

  If the file is not overlaid, SIPP does not print the segment column. Below the header, SIPP displays the segment, base, and offset you have specified by responding to the dialog prompts.

  A sample dialog format follows. In this example, SIPP is to begin code modifications in overlay segment 2 of program PROG.SAV.

  ```
  .R SIPP RET
  *PROG=PROG RET
  Segment? 2 RET
  Base?    20000 RET
  Offset?  100 RET

  Segment     Base     Offset     Old     New?
  000002      20000    20100   103425
  ```

  Under the column marked *Old*, SIPP displays the contents of the currently open location. Under the column marked *New*?, you can enter either a new value for the current location and/or a command. The SIPP Command Descriptions section gives more details on opening and modifying locations. Table 24–2 summarizes the commands you can enter.

NOTE

SIPP does not make changes to a file as you type them. Instead, SIPP stores the changes in a buffer, allowing you to abort a partially completed patch operation without leaving behind a partially patched file. When you finish a patching operation by pressing CTRL/Y or CTRL/Z several times (see Table 24–2), SIPP makes all the changes in one pass.

SIPP Command Summary

Table 24–2 summarizes the commands you can enter during the code modification procedure and lists the sections in which you can find more details on each command. You can enter a command by pressing either LF (line-feed key) or RETURN.

Table 24–2: SIPP Commands

Command	Function
RET or LF	Closes the current location without modifying it, and opens and displays the next location.
n RET	Enters the value represented by n in the current location, closes it, and opens the next location.
^ RET	Closes the current location without modifying it, and opens the previous location.
n^ RET	Enters the value represented by n in the current location, closes it, and opens the previous location.
\ RET	Reopens the current location as a byte (starting with the low, or even, byte for that word). From this point, SIPP will continue opening byte locations and accepting byte values. Do not use this command when in Radix–50 or ASCII mode.
/ RET	Reopens the current location as a word. SIPP displays the contents of the currently open word location. All further displays and input will be word values. Do not use this command when in Radix–50 or ASCII mode.
;O RET	Reopens the current location as an octal word value. This is the default mode. Use ;O to return to octal after having been in Radix–50 or ASCII mode. All further displays and input are octal values.
;A RET	Displays the byte of the current location as an ASCII value. All further displays are in ASCII, and SIPP advances in byte mode.
;Ax RET	Inserts an ASCII character represented by x in the byte of the current location, closes that byte, and opens and displays the next location. Use this command for inserting only one ASCII character at a time. You can also use this command to search for an ASCII value (see the description of the ;S command, *Searching Through Files*).
;R RET	Displays the current location as a word of up to three Radix–50 characters. All further displays are in Radix–50.
;Ryyy RET	Inserts up to three Radix–50 characters represented by yyy into the current location. SIPP then closes the current location, and opens and displays the next location. Use this command for inserting up to three Radix–50 characters. You can also use this command to search for a Radix–50 value (see the description of the ;S command, *Searching Through Files*).

Table 24–2 (Cont.): SIPP Commands

Command	Function
;S RET	Searches for a value within the file. When you type this command, SIPP prompts you for a value for which it is to search. SIPP also prompts you for the boundaries within which you want it to conduct the search.
;V RET	Displays all the modifications you have made in the current patching session. You can use this command at any time, except in response to Checksum?.
CTRL/Z RET	Backs up to the previous prompt: Offset?, Base?, or Segment?. This command allows you to insert code modifications in more than one area of the file during the same patching session.
CTRL/Y RET	Completes the current patching session, installs the patch, creates the command file (if requested), and prompts you with an asterisk for another file specification.

SIPP Command Descriptions

Opening and Modifying Locations Within a File (RET, LF, & n)

After you guide SIPP to the location in the file where you want to begin making code modifications, SIPP displays a header under which it lists the address and contents of the location specified, called the current location. Under the last column in the header, *New?*, you can enter a value that replaces the contents of the current location. After you enter the new value, press RETURN to advance to the next location.

In the following example, the value 240 is inserted in the current location. Next, a RETURN advances SIPP to the next 16-bit location.

```
Base          Offset        Old      New?
 001000         001200    004767      240 RET
 001000         001202    003106
```

If you do not want to modify the current location, simply press RETURN to advance to the next location.

Backing Up Through Files (^)

When you type the up-arrow character (^) followed by a RETURN in place of entering data into the current location, SIPP closes the current location and opens the previous location. If you specify a value followed by an up-arrow, SIPP enters that value into the current location, closes that location, and opens the previous location.

If you type an up-arrow when the offset from a specified base is 0, SIPP opens the previous location and displays the offset as a double-precision negative number.

In the following example, the value 112000 is entered into the current location, and the previous location is opened.

```
Base        Offset       Old      New?
 002000       002134   020027     112000^  RET
 002000       002132   001732
```

In the last example, notice how SIPP decrements the offset by 2 to designate the previous 16-bit location.

You can use the up-arrow after you use the backslash (\) to back up to the previous byte (if currently in word mode). You can also use the up-arrow after you use the slash (/) to back up to the previous word (if currently in word mode).

Advancing in Bytes (RET or \)

By default, SIPP operates in word mode. That is, locations are displayed as 16-bit words, and values are displayed and entered as word values. If you type the backslash character (\), SIPP closes the current location and reopens the low, or even byte, of that location. Values that are displayed and entered from this point are in bytes.

To revert to word mode, type the slash character (/). When you type the slash, SIPP reopens the current location as a 16-bit word.

The following example uses the backslash to advance in byte mode, and then the slash to revert to word mode. Notice that SIPP displays a new header each time it changes from word mode to byte mode, and vice versa.

```
Base        Offset       Old     New?
 0020000     002112   003002     \ RET

Base         Offset      Old     New?
 002000       002112   002        RET
 002000       002113   006        RET
 002000       002114   132       / RET

Base         Offset      Old     New?
 002000       002114   003132
```

If you are in byte mode when you get the Offset? prompt, SIPP automatically resets itself to word mode.

Entering Octal Values (;O)

Use the ;O command, followed by a RETURN, to reopen the current location, and display its contents as an octal value. Since octal mode is the default setting, you need to use it only if you are currently operating in ASCII or Radix–50 mode and wish to revert to octal mode. You can also use the ;O command to switch from byte mode to word mode.

The following example uses the ;O command to switch from ASCII mode to octal mode.

```
Base       Offset        Old      New?
 002000     002100    051101      ;A RET
 002000     002100       <A>      ;O RET

Base       Offset        Old      New?
 002000     002100    051101
```

Note that unlike the ;A and ;R commands, ;O accepts no optional argument. If you return to the *Offset?* prompt, SIPP automatically resets itself to octal mode.

Displaying and Entering ASCII Values (;A or ;Ax)

Use the ;A command, followed by a RETURN, to open the current location as a byte and display its contents as an ASCII value. When you use the ;A command, SIPP continues to display contents in ASCII until you use the ;O or ;R command. Note that when you operate in ASCII mode, you advance through the file in byte mode.

The following example uses the ;A command to open the low byte of the current location and display its contents as an ASCII value.

```
Base        Offset       Old     New?
 003000      003100   050524     ;A RET
 003000      003100      <T>      RET
 003000      003101      <Q>
```

Use the ;Ax command to insert an ASCII character, represented by x, into the low byte of the current location. When you use the ;Ax command, SIPP enters the ASCII character directly into the current byte, closes that byte, opens and displays the next location as an octal, ASCII, or Radix–50 value (depending on what mode you were in prior to using the ;Ax command). Note that you can insert only one ASCII character at a time, and that you should not insert control characters. You can use the ;Ax command when displaying in ASCII mode.

The next example uses the ;Ax command to enter the ASCII character, W, into the current byte and proceed to the next byte.

```
Base        Offset       Old     New
 003000      003100   050524     ;AW  RET
 003000      003101   121
```

You can also use the ;Ax command to search for an ASCII value (see the description of the ;S command, *Searching Through Files*).

Displaying and Entering Radix–50 Values (;R or Ryyy)

Use the ;R command, followed by a RETURN, to reopen the current location and display its contents in Radix–50. When you use the ;R command, SIPP continues in Radix–50 mode until you use either the ;A or ;O command. Note that Radix–50 mode advances in word mode.

The following example uses the ;R command to reopen the current location and display its contents as a Radix–50 value.

```
Base        Offset       Old     New?
 001000      005220   071070     ;R  RET
 001000      005220   <RK>        RET
 001000      005222   <TES>
```

If the contents of a location is an invalid Radix–50 value, SIPP displays the contents as <???>.

You can use the ;Ryyy command to insert up to three Radix–50 characters, represented by yyy, into the current location. When you use the ;Ryyy command, SIPP inserts the Radix–50 value into the current location, closes the current location, and opens and displays the contents of the next location as an octal, ASCII, or Radix–50 value (depending on what mode you were in prior to using the ;Ryyy command).

If you use the ;Ryyy command, and you enter only two Radix–50 characters, SIPP inserts a blank as the third character. Likewise, if you enter only one Radix–50 character, SIPP inserts blanks for the second and third characters. If you use an imbedded blank (for example, X Z), SIPP inserts all characters as typed. Note that you can insert up to only three Radix–50 characters at a time. Use the ;Ryyy command only when the low byte of the current location is open; SIPP displays an error message if you attempt to insert a Radix–50 value when the high byte of the current location is open.

The following Radix–50 values are valid for use with the ;Ryyy command:

A through Z
0 through 9
$
*
.
%

Note that a space is also a valid Radix–50 character, and that SIPP translates the percentage character to a dot (.).

The following example uses the ;Ryyy command to insert three Radix–50 characters in the current location, and proceed to the next location.

```
Base         Offset        Old    New?
 001000       005332    000240    ;RABC RET
 001000       005334    002110
```

You can also use the ;Ryyy command to search for a Radix–50 value (see the description of the ;S command, *Searching Through Files*).

Searching Through Files (;S)

You can use the ;S command to search between two specified boundaries of a file for a given value. With this feature, you can find the location where you want to make a change by searching for a specific value.

Search Procedure:

1. To request a search, type the following in response to any of SIPP's dialog questions or in place of entering new data in the current location.

   ```
   ;S
   ```

   Do not type the ;S command in response to the *Checksum?* prompt.

   After you type the ;S command, SIPP responds with the following prompt:

   ```
   Search for?
   ```

2. Respond to the search prompt by entering the value for which you want SIPP to search. You can use the ;Ax or ;Ryyy command to search for ASCII or Radix–50 values. If you type a backslash after the value you enter, SIPP searches for a byte value. Otherwise, it searches for a word value. Note that if you use the ;Ax notation to search for an ASCII value, SIPP conducts the search in byte mode.

   SIPP then asks for the lower address limit at which to begin the search:

   ```
   Start?
   ```

3. Respond to the lower-address prompt by entering an address, followed by a RETURN, or just a RETURN. If you enter only a RETURN, SIPP begins its search at the beginning of the file. If you enter an address, SIPP begins the search at that address.

If you are searching through an overlay segment, use the following notation for the start address:

```
n:m
```

In the n:m notation, n specifies the number of the segment you want to search, and m specifies the offset from the start of the segment where you want SIPP to begin the search.

SIPP then asks for the upper-address limit for the search:

```
End?
```

4. Respond to the upper-address prompt by entering an address (including the n:m notation) or a RETURN. If you only press RETURN, SIPP searches to the end of the file or volume. (If you use the /A option, SIPP searches to the end of the last block in the file or volume; otherwise it searches up to and including the last address in the program.) If you enter an address, SIPP conducts the search up to, but not including, that address.
5. After you have specified the search limits, the search begins. Each time SIPP finds a value that matches the one you specified, SIPP displays the address of that value. If you use the /A option or if SIPP is searching the root segment of a program, SIPP displays the search results as follows:

```
Found at nnnnnn
```

If a search crosses segments in an overlaid file, SIPP displays the following each time it finds the specified value:

```
Found at seg:mmm,nnn
```

In the *seg:mmm,nnn* notation:

seg	specifies the segment number
mmm	specifies the load map address of the segment
nnn	specifies the offset from the start of the specified segment

Note that if you are searching an overlaid file, and you have specified the /A option in the command line, SIPP does not use the *seg:mmm,nnn* notation.

Verifying Your Modifications (;V)

Use the ;V command to list at the terminal all the changes you have made during the current patching session. After SIPP displays the addresses and new contents of all the locations that have changed, SIPP returns you to the operation that was interrupted.

You can use the ;V command at any time, except in response to the *Checksum?* prompt and the search *Start?*, and *End?* prompts. You can use the ;V command in response to the *Search for?* prompt.

The following example uses the ;V command to list at the terminal all the changes that have been made during the current patching session.

```
Base         Offset        Old    New
 003000       003200     003112   240  RET
 003000       003202     002300   RET
 003000       003204     002300   RET
 003000       003206     000230   240  RET
 003000       003210     000101   ;V  RET

Base         Offset        Old    New?
 003000       003200     003112   000240
 003000       003206     000230   000240

Base         Offset        Old    New?
 003000       003210     000101
```

Note that when you use the ;V command to verify your modifications, all displays are in octal words. Note also that if you change a location and later restore that location to its original contents, SIPP includes that location in the verification.

Backing Up to a Previous Prompt (CTRL/Z RET)

You can use the CTRL/Z sequence (or up-arrow Z), followed by a RETURN, to back up to a previous prompt. For example, after you have modified a series of locations, you can press CTRL/Z RETURN to back up to the *Offset?* prompt. Backing up to a previous prompt enables you to examine and/or modify other series of locations in your program.

- If you press CTRL/Z in place of entering a value in a location, SIPP prompts "*Offset?*".
- If you press CTRL/Z RETURN in response to "*Offset?*", SIPP prompts "*Base?*".
- If you press yet another CTRL/Z RETURN, SIPP does one of two things:
  — Prompts "*Segment?*", if the file is overlaid.
  — Prompts you for a checksum (if you used /C), then installs the patch (if the checksum is valid). (Note that if you have used the /L option, SIPP does not install the patch, but does create the command file, if requested.)

If the file is overlaid, and you press CTRL/Z again followed by a RETURN sequence in response to *Segment?*, SIPP prompts you for a checksum (if specified) then installs the modifications.

Using CTRL/Y provides a more efficient way of installing a patch (see the following subsection). The CTRL/Z sequence is designed primarily to request a particular prompt.

Completing Code Modifications (CTRL/Y RET)

You can type the CTRL/Y sequence (or up-arrow Y), followed by a RETURN, to install the code modifications you have entered. If you have used the /C option, which requires you to enter a checksum, SIPP will prompt you for a checksum before it installs the modifications. If the checksum you type is valid, SIPP then installs the patch. If you have used the /L option and you enter the correct checksum, SIPP does not install the patch, but does create the command file, if requested.

After SIPP installs the modifications, an asterisk appears in the left margin, indicating that SIPP is ready to accept a new command string.

Extending Files and Overlay Segments

The limits to which you can extend programs and overlay segments while patching vary, depending on whether the program is:

- Nonoverlaid
- Overlaid, but has only low-memory overlays
- Overlaid, but has only extended-memory overlays
- Overlaid, and has both low-memory and extended-memory overlays

The following sections each respectively describe how to deal with the preceding types of programs. These sections describe in detail:

- The restrictions on extending programs, root sections, and overlay segments
- What data within your program SIPP does or does not automatically modify as you extend root sections and/or overlay segments.

  The following table lists the data that SIPP may automatically modify as you make extensions. Note that each item is identified by an abbreviation; the subsections that follow reference these items by these same abbreviations.

Possible SIPP Data Changes

Data That Can Change	Description
Location 50	Contains the last address used by the program, if the program is nonoverlaid. If the program has low-memory overlays, location 50 contains the last address used by the low-memory overlay region(s). If the program has only extended-memory overlays, location 50 contains the last address used by the root.
Region Size	Indicates the size of the extended-memory region. This data appears in the extended-memory overlay handler.
High Root + 2	Indicates the address of the next available location beyond the root segment. This data appears in either the low-memory overlay handler or the extended-memory handler.
High /O + 2	Indicates the address of the next available location beyond the last low-memory overlay region. This data appears in either the low-memory overlay handler or the extended-memory overlay handler.

Data That Can Change	Description
Word Count in Segment	Indicates the number of words in the current overlay segment. This data appears in the overlay handler segment table.
Window Definition Block Size and Length	Indicates the window size and length to map in the window definition block (WDB) for the extended-memory overlay you are extending. This data appears in each extended-memory overlay segment's WDB.
Window Definition Block Offset	Indicates the offset into the extended-memory region of the windows following the segment you are extending. This data appears in each extended-memory overlay segment's WDB.

Extending a Nonoverlaid Program

If necessary, SIPP automatically extends a nonoverlaid program up to the end of the last block of the save image on which the program exists (SIPP automatically modifies location 50). See DUP (Chapter 7) for details on extending a nonoverlaid program beyond that point.

Extending a Program with Low-Memory Overlays Only

If your program is overlaid, but has only low-memory overlays, SIPP does not permit you to extend the root. If an overlay segment is not in the last overlay region, you can extend it to the size of the largest segment in its region. If the overlay segment is in the last overlay region, you can extend it to the end of the last block of that overlay segment.

The following table shows the locations that SIPP modifies if necessary when you extend the root or overlay segment of a program that has low-memory overlays only. Note in this table that there is a column heading for an overlay segment that is not the largest in its overlay region (*Not Largest in Region*), and for an overlay segment that you extend beyond the largest overlay segment in its region (*Past Largest in Last Region*). YES indicates that SIPP does modify the data in question if necessary, NO indicates SIPP does not modify the data in question, and N/A indicates that the data in question is not applicable.

Segment Limits for Only Low-Memory Overlays

Data That Can Change	Not Largest in Region	Past Largest in Last Region
Location 50	NO	YES
Region Size	N/A	N/A
High Root + 2	NO	NO
High /O + 2	NO	YES

Data That Can Change	Not Largest in Region	Past Largest in Last Region
Word Count in Segment	YES	YES
Window Definition Block Size and Length	N/A	N/A
Window Definition Block Offset	N/A	N/A

Extending a Program with Extended-Memory Overlays Only

If your program is overlaid, but has extended-memory overlays only, you can extend the root segment up to the end of the last block on which the root resides. You can also extend any overlay segment up to the end of the last block of that particular segment, so long as you do not exceed the physical address space.

The following table shows the locations that SIPP modifies if necessary when you extend the root or overlay segment of a program that has extended-memory overlays only. Note in this table that there is a column heading for the root (*Root*), an overlay segment that is not the largest in its overlay region (*Not Largest*), and an overlay segment that you extend beyond the largest overlay segment in its region (*Past Largest*). YES indicates that SIPP does modify the data in question if necessary, and NO indicates SIPP does not modify the data in question.

Segment Limits for Only Extended-Memory Overlays

Data That Can Change	Root	Not Largest	Past Largest
Location 50	YES	NO	NO
Region Size	NO	YES	YES
High Root + 2	YES	NO	NO
High /O + 2	YES	NO	NO
Word Count in Segment	NO	YES	YES
Window Definition Block Size and Length	NO	YES	YES
Window Definition Block Offset	NO	YES	YES

Extending a Program with Low- and Extended-Memory Overlays

If your program has both low-memory and extended-memory overlays, SIPP does not permit you to extend the root segment. You can extend any low-memory overlay segment up to the size of the largest segment in the same region. If the low-memory overlay segment is in the last low-memory overlay region, you can extend it to the end of the last block of that overlay segment.

Extending Files and Overlay Segments

You can extend any extended-memory overlay segment up to the end of the block limit of that particular segment, so long as you do not exceed your physical address space.

The following table shows the locations that SIPP modifies if necessary when you extend the root or overlay segment of a program that has both low-memory and extended-memory overlays. Note in this table that there is a column heading for a low-memory overlay segment that is not the largest in its overlay region (*/O Not Largest*), a low-memory overlay segment that extends beyond the largest segment in its region (*/O Past Largest*), an extended-memory overlay segment that is not the largest in its region (*/V Not Largest*), and an extended-memory overlay segment that extends beyond the largest segment in its region (*/V Past Largest*). YES indicates that SIPP does modify the data in question if necessary, and NO indicates SIPP does not modify the data in question.

Segment Limits for Both Low- and Extended-Memory Overlays

Data That Can Change	/O Not Largest	/O Past Largest	/V Not Largest	/V Past Largest
Location 50	NO	YES	NO	NO
Region Size	NO	NO	YES	YES
High Root + 2	NO	NO	NO	NO
High /O + 2	NO	YES	NO	NO
Word Count in Segment	YES	YES	YES	YES
Window Definition Block Size and Length	NO	NO	YES	YES
Window Definition Block Offset	NO	NO	YES	YES

Note: It is possible, when extending extended-memory overlay segments, to exceed the 96K-word physical-address space limit (SIPP displays a warning message if you do this). If you do exceed the 96K-word limit, press CTRL/C to abort the patching session; many system-communication area locations will have already changed to contain invalid data.

Verifying Your Work with Checksum

SIPP's checksum algorithm creates the checksum only after you have finished creating a patch. The checksum helps you verify your work. It lets you compare the patch you make to another that is known to be correct. The checksum does not tell you where your error is, but it does tell you that an inconsistency exists. SIPP's checksum algorithm uses the calculated address of a changed location and its new contents. SIPP includes in its checksum only those values of locations that have changed during a patching session.

If you change a word several times during a patching session, SIPP enters into its checksum only the last value you specify. This feature allows you to correct a mistake, yet maintain a valid checksum.

If you are creating a checksum (/D option), SIPP displays the following when you finish the patch:

```
Checksum=nnnnnn
```

If you are verifying a checksum (/C option), SIPP asks the following when you complete the patch:

```
Checksum?
```

Respond by entering the checksum for the patch.

If the checksum is incorrect, SIPP displays:

```
?SIPP-E-Checksum error
```

SIPP then returns to the beginning of its dialog (the *Segment?* or *Base?* prompt), allowing you to find and correct your error without exiting from the patching session. In this way, you do not lose the changes you have made; you can go back and verify them (see the command description on *Verifying (;V)* for details on the verify command).

SIPP does not install the patch until you enter the correct checksum. You can press CTRL/C to abort the patching procedure.

Running SIPP from a Command File

The SIPP command file contains the commands necessary to install a patch in a particular file or volume. The order in which the modifications appear in the command file may not correspond to the actual sequence in which you typed them; however, the changes are the same as you typed. The contents of the command file always appear as octal word values. When you specify a command file in the initial command string, SIPP creates that file for use as a command file. If you use the /L option when you create the command file, SIPP installs the patch contained within only when you run this file as a command file. SIPP assigns this file a COM default file type.

A command file always contains a checksum generated during the input session. If you use the /C option, SIPP prompts you for a checksum after you finish making the code modifications. If the checksum is valid, SIPP completes the command file, and you can execute it at any time. If you use the /A option, SIPP inserts /A in the command file.

The command file TEST.COM is created in the following example. Note that SIPP does not modify TEST.SAV in the example, because /L was specified in the command string:

```
.R SIPP  RET
*TEST=TEST/L  RET
Base?   5000  RET
Offset? 20  RET

Base         Offset        Old     New
 005000       000020     032764    240 RET
 005000       000022     177400    240 RET
 005000       000024     000002    1016 RET
 005000       000026     001016     CTRL/Y  RET
* CTRL/C
```

A copy of TEST.COM as it appears in command-file format follows. The number 165617 (the last line in the file) is the checksum for that patch:

```
.TYPE TEST.COM
RUN SIPP
DK:TEST.SAV/C
5000
20
240
240
1016
^Y
165617
^C
```

Run the command file TEST.COM as you would run any command file:

```
.@TEST  RET
```

If you run a SIPP command file when the SET TT: QUIET setting is in effect, SIPP overstrikes its output at the terminal but does install the patch correctly.

Running SIPP from a Batch Stream

An easy way to install a patch from a BATCH stream is to follow the instructions for creating a command file. When you get your command file, simply open it with an editor, enter the BATCH commands, and insert a dot before the line RUN SIPP, and insert asterisks before each subsequent line. Remember to remove the CTRL/C (^C) from the command file. An example of preparing TEST.COM (from the previous section) for a BATCH stream follows.

```
$JOB/RT11
        TTYIO
.RUN SIPP
*DK:TEST.SAV/C
*5000
*20
*240
*240
*1016
*^Y
*165617
$EOJ
```

Chapter 25

Source Language Patch Utility (SLP)

The Source Language Patch Utility (SLP), is a patching tool you can use for maintaining source files that exist on an RT–11 device.

SLP accepts as input a source file you wish to patch and a command file that you create when you compare two source programs using the source compare program, SRCCOM, described in Chapter 28. When you use SLP along with the SRCCOM command file, you can quickly and easily patch one version of a source program to match another version. Chapter 28 also describes the procedure you can use to create a patch command file that is suitable for input to SLP.

Calling and Terminating SLP

To call SLP from the system device, respond to the dot (.) displayed by the keyboard monitor by entering the following:

```
.R SLP RET
```

The Command String Interpreter (CSI) prints an asterisk (*) at the left margin of the terminal and waits for a command string. If you only press RETURN in response to the asterisk, SLP prints its current version number.

You can press CTRL/C to halt SLP and return control to the monitor when SLP is waiting for input from the terminal. To restart SLP, enter R SLP or REENTER in response to the monitor's dot.

Command-Line Syntax

Enter a command line according to this general syntax:

[outfil][,listfil]=infil,comfil/[option...]

where:

outfil	specifies the updated source file. The default file type is MAC.
listfil	specifies the listing file. When you specify this file, SLP creates a numbered listing of the updates SLP made to the source file. The default file type is LST.
infil	specifies the source file you want SLP to update. The default file type is MAC.
comfil	specifies the command file that contains the commands for updating the source file. The default file type is MAC. You can create this file by specifying a SLP-filespec in a SRCCOM command line. A SLP input file created by SRCCOM has the file type SLP.
/option	specifies one of the options listed in Table 25–1.

Although either output file can be omitted, you must use one or both.

SLP Option Summary

Table 25–1 lists the options you can use in the command line.

Table 25–1: SLP Options

Option	Function
/A	Disables audit-trail generation. The audit trail is a string of characters that SLP appends to the end of each updated line in the output files. The audit trail keeps track of the update status of each line in the output file. You can use the /A option if you do not want SLP to use the audit trail in both the updated source file and the listing file.
/B	Inserts spaces instead of tabs between the source line and the audit trail.
/C[:n]	Determines or validates the contents of the SLP input file, SLP command file, or both, by using checksums. Use /C to determine the checksum of a file. Use /C:n to verify the contents of a file. SLP computes the checksum for the file, and compares the result to the value you specify with n.
/D	Creates a double-spaced listing. When you use this option, SLP double spaces between the lines in a listing file.
/L:n	Specifies the size of the source line, where n represents the maximum number of characters you want in the source line. The default buffer size for formatting lines is 200_{10} bytes. If you expect the size of the command lines or source lines to be greater than what can fit in the line buffer, you can use this option to change the buffer size. SLP interprets the number you specify for n as an octal number; if you enter a decimal number, use a decimal point. The line buffer must be at least as long as the sum of the column number where the audit trail begins and the number of characters in the audit trail.
/N	Suppresses the creation of a backup file when SLP updates the input file.
/P:n	Specifies the start column of the audit trail, where n represents the column number in which you want the audit trail to start. If the number you specify for n is decimal, be sure to use a decimal point after the number. By default, SLP starts the audit trail in column 73_{10}. If a source line extends beyond the column where the audit trail begins, the audit trail can overstrike the source line. If you use the /P:n option, you start the audit trail in any tab stop column. SLP rounds up the number you specify to the nearest tab stop column. If, for example, you specify 46 for n, SLP rounds this number to 49.

Table 25–1 (Cont.): SLP Options

Option	Function
/S:n	Specifies size of the audit trail, where n represents the number of characters you want in the audit trail. If the number you specify is decimal, be sure to use a decimal point after the number. The default number of characters in the audit trail is 11_{10}. The maximum number of characters you can specify for the audit trail is 16_{10}.
/T	Retains trailing blanks and tabs in the input source file. By default, SLP removes spaces and tabs that appear at the end of lines in the input source file.

NOTE

SLP is not accessible through DCL commands.

Example Use of SLP

Note the following two sample text files CAESAR.MAC and ANTONY.MAC:

- **CAESAR.MAC**

```
FRIENDS, ROMANS, COUNTRYMEN!
LEND ME YOUR EARS!
I COME TO BURY CAESAR,
NOT TO PRAISE HIM.
THE EVIL THAT MEN DO
LIVES AFTER THEM.
THE GOOD IS OFT INTERRED
WITH THEIR BONES;
SO LET IT BE WITH CAESAR!
```

- **ANTONY.MAC**

```
FRIENDS, ROMANS, COUNTRYMEN!
LEN ME YOUR EARS!
I COME TO BURY CAESAR,
NOT TO PRAISE HIM.
THE EVIL THAT MAN DO
LIVES AFTER THEM.
THE GOOD IS OFT ENTERED
WIT THEIR HOMES;
SO LET IT BE WITH CAESAR!
```

Notice the following differences between the two preceding files.

Line	ANTONY.MAC	CAESAR.MAC
2	LEN	LEND
5	MAN	MEN

Line	ANTONY.MAC	CAESAR.MAC
7	ENTERED	INTERRED
8	HOMES	BONES

This section uses SLP to patch the text file, ANTONY.MAC, so that it matches the file CAESAR.MAC. By specifying a SLP-filespec in a SRCCOM command line, you can get an appropriate command file (for example, CAESAR.SLP) to make both preceding files alike. See Chapter 28 for instructions on how to automatically create a SLP command file.

The following command line directs SLP to patch ANTONY.MAC so that it matches CAESAR.MAC.

```
.R SLP RET
*ANTONY,ANTONY=ANTONY,CAESAR.SLP RET
```

When executing the preceding command, SLP:

- Updates the source file ANTONY.MAC, making it identical to CAESAR.MAC.
- Assigns a BAK file type to the input file ANTONY.MAC.
- Assigns a MAC file type to the updated source file.
- Creates a listing file ANTONY.LST that lists each line by number and appends an audit trail to each new line.

ANTONY.LST is as follows:

```
SLP V05.05

    1.  FRIENDS, ROMANS, COUNTRYMEN!
    2.  LEND ME YOUR EARS!           ;**NEW**
    3.  I COME TO BURY CAESAR,       ;**-1
    4.  NOT TO PRAISE HIM.
    5.  THE EVIL THAT MEN DO         ;**NEW**
    6.  LIVES AFTER THEM.            ;**-1
    7.  THE GOOD IS OFT INTERRED     ;**NEW**
    8.  WITH THEIR BONES;            ;**NEW**
    9.  SO LET IT BE WITH CAESAR!    ;**-2
```

Note that when SLP updates a line, it appends an additional audit trail below the audit trail of the updated line. The additional audit trail keeps track of the number of consecutive lines that have been updated. In ANTONY.LST, above, note the audit trails ;**-1 and ;**-2.

Determining and Validating the Contents of a File

Use the checksum option (/C[:n]) to determine or validate the contents of a file. The checksum option directs SLP to compute the sum of all ASCII data in a file. If you specify the command in the form /C:n, /C directs SLP to compute the checksum and compare that checksum to the value you specify as n.

To determine the checksum of a file, enter the SLP command line with the /C option applied to the appropriate file (the file whose checksum you want to determine). For example:

```
INFILE,INFILE=INFILE.MAC/C,INFILE.SLP
```

SLP responds to this command with a message in the following format:

```
?SLP-I-DEV:FILNAM.TYP checksum is n
```

SLP generates a similar message when you request the checksum for the command file.

To validate the changes made to a file, enter the checksum option in the form /C:n. SLP compares the value it computes for the checksum with the value you specify as *n*. If the two values do not match, SLP enters no changes and displays a message reporting the checksum error as either a source file or a correction file checksum error, whichever is appropriate.

```
?SLP-F-Source file checksum error
```

or

```
?SLP-F-Correction file checksum error
```

Checksum processing always results in a nonzero value.

Do not confuse this checksum with the record checksum byte.

Creating and Maintaining a SLP Command File

SLP is a line-oriented patching tool. That is, you make changes to entire lines, not to individual characters or strings of characters within a line. If you want to change only a few characters within a line, you must enter a new line.

Although Digital recommends that you create the SLP input command file by specifying a SLP-filespec in a SRCCOM command line, you can use any RT–11 editor to create it yourself.

The next section describes the commands, or operators, you use to create that command file. This procedure is tedious, however, and in most cases unnecessary. But, for completeness, the procedure is included with this chapter. Table 25–2 lists the commands, or operators, you enter into the command file. Subsequent sections in this chapter describe various ways you can use the SLP utility to change lines in a file.

Table 25–2: SLP Command-File Operators

Operator	Function
–	Indicates the start of an update. SLP ignores any data that precedes this operator in a SLP command file. If SLP finds characters before this operator in a command file, SLP prints a warning and the characters are ignored. If no operator of this type is found in a command file, SLP prints an error message and the CSI prompt (*) appears.
\	Disables the audit trail. Note that this operator must appear on a line by itself. If it appears in the first column of a line with additional information following it, the audit trail is disabled and the rest of the command line is ignored. If used in any column other than column one, a syntax error occurs.
%	Enables the audit trail.
/	Indicates the end of an update or a series of updates; it appears as the last character in the command file.
//	Indicates the end of one of a series of update texts in a single command file; each text updates one input file. This operator is used when you want to include updates for more than one file in a single command file. Enter the double slash (//) on a line by itself after each update text in the series. Then on the next line enter the command line that specifies the next input file to be updated, and on the succeeding lines the update text for that file. Note that the command file specified in each command line must be the same as the command file specified on the first command line.

Table 25–2 (Cont.): SLP Command-File Operators

Operator	Function
<	Serves as an escape character for characters SLP would otherwise interpret as operators. For example, if you want to include a slash (/) in a source file, type the less than character (<) before the slash. Then, SLP will not interpret the slash as an operator. You can use the less than character as an escape character for all SLP command file operators.

Formatting the Update Line in a SLP Command File

The following is the general format of the SLP command-file update line with input line(s):

```
–locator1,[locator2],[/audit trail/][;]
input line
   .
   .
   .
```

where:

–	indicates that this is an update line.
locator1	specifies a character string that serves as a line locator. SLP moves the line pointer to the line specified by the line locator. You can specify this line locator with any of the locator forms described below.
locator2	specifies a character string that, when used with locator1, defines the end of a range of lines you want to delete or replace. You can specify this line locator with any of the locator forms described below. You cannot define a range of lines in a backwards direction; the line referenced by locator2 must occur in the source file after the line referenced by locator1.
/audit trail/	specifies a character string you use as an audit trail. SLP appends the audit trail to the right of each updated line. You must delimit the audit trail with slashes (/).
inputline	specifies a line of new text that SLP inserts into the file immediately following the current line. You can enter as many input lines as you want.
;	is an optional command-line terminator.

All fields in the update line are positional. That is, if you specify only *locator1* and *audit trail*, you must use two commas between those two fields. If you want to specify only the *audit trail*, you must precede the audit trail with two commas.

The update lines in a command file must edit the source file in a forward direction, from beginning to end. Each locator1 must point to a line that appears in the source file before the lines pointed to by any succeeding locator1.

The line locators can take one of the following forms:

```
/string/[+n]
/string...string/[+n]
number[+n]
.+n
```

Table 25–3 describes these locators.

Table 25–3: Descriptions of SLP Line Locators

Locator	Description
/string/[+n]	specifies an ASCII character string. You must delimit any string you enter with slashes. SLP locates the first occurrence of this string, and moves the line pointer to the line that contains that string. +n represents the offset from the line that contains the string. You must use the plus character (+) with the n notation.
/string...string/[+n]	specifies an ASCII character string. SLP locates the line in which the two strings delimit a larger string. Use the ellipsis (...) in this locator form to separate the two strings. +n represents the offset from the line specified by the string...string locator.
number[+n]	specifies the line number to which SLP is to move the line pointer. +n represents the offset from the line specified by number.
.+n	specifies the offset from the current line pointer. SLP interprets the period (.) as the current line pointer location, and the +n as the offset from it. You must use the plus character (+).

Creating a Numbered Listing

You can use SLP to create a numbered listing of the input source file. In creating a command file, you should use a numbered listing when you prepare command input. To generate a numbered listing, enter the following lines:

```
.R SLP RET
*,listfile=infile RET
```

Listfile represents the listing file SLP produces, and infile represents the input source file. Here is a file, PROG.MAC, from which SLP is to create a numbered listing:

```
        .TITLE   PROG.MAC   VERSION 1
        .MCALL   .TTYOUT, .EXIT, .PRINT

EXP:    .PRINT   #MESSAG
        MOV      #N,R5
FIRST:  MOV      #N+1,R0
        MOV      #A,R1
```

The following command line creates a numbered listing, PROG.LST, of the preceding file, PROG.MAC:

```
*,PROG=PROG RET
```

After SLP processes the command above, it produces the following listing of PROG.MAC:

```
SLP  V05.05

     1.          .TITLE   PROG.MAC   VERSION 1
     2.
     3.          .MCALL   .TTYOUT, .EXIT, .PRINT
     4.
     5.  EXP:    .PRINT   #MESSAG
     6.          MOV      #N,R5
     7.  FIRST:  MOV      #N+1,R0
     8.          MOV      #A,R1
```

Adding Lines to a File

To add lines to a file, enter in the command file one of the following three locator forms:

- -number
- -.[+n],,[/audittrail/]
- -/string/

Notice in the second locator form the two commas between the locator and the audit trail. You do not have to insert these commas if you are not specifying an audit trail.

Consider the following file, NUMBER.PAS:

```
PROGRAM NUMBER;

TYPE     TEXT      =FILE OF CHAR;
         PTR       =^WORDNODE;
         WORDNODE=RECORD
                    WORD:ARRAY[1..30] OF CHAR;
                    NEXT:PTR;
                    END;

VAR      P,TOP      :PTR;
         INTEXT     :TEXT;
         I          :INTEGER;
```

To add lines to the preceding file, we have the command file, OMEGA.MAC: This file instructs SLP to insert the line `(*POINTER TO NODE*)` between the fourth and fifth lines of NUMBER.PAS:

```
-/PTR/
(*POINTER TO NODE*)
/
```

When SLP processes OMEGA.MAC with NUMBER.PAS, it produces the following updated listing file.

```
*NUMBER.PAS,NUMBER=NUMBER.PAS,OMEGA.MAC  RET

SLP V05.05

 1.  PROGRAM NUMBER;
 2.
 3.  TYPE     TEXT    =FILE OF CHAR;
 4.           PTR     =^WORDNODE;
 5.  (*POINTER TO NODE*)                              ;**NEW**
 6.           WORDNODE=RECORD
 7.                    WORD:ARRAY[1..30] OF CHAR;
 8.                    NEXT:PTR;
 9.                    END;
10.
11.  VAR      P,TOP    :PTR;
12.           INTEXT   :TEXT;
13.           I        :INTEGER;
```

SLP has numbered the lines, inserted the new text, and appended the default audit trail (`;**NEW**`) to the new line.

The next example uses the same source file, but uses the following command in the command file, SIGMA.MAC:

```
-/WORDNODE=/+2
               ID  :INTEGER;
/
```

When SLP processes SIGMA.MAC with the source file NUMBER.PAS, it generates the following listing file:

```
*NUMBER.PAS,NUMBER=NUMBER.PAS,SIGMA.MAC RET

SLP V05.05

 1.  PROGRAM NUMBER;
 2.
 3.  TYPE      TEXT     =FILE OF CHAR;
 4.            PTR      =^WORDNODE;
 5.            WORDNODE=RECORD
 6.                     WORD:ARRAY[1..30] OF CHAR;
 7.                     NEXT:PTR;
 8.                     ID  :INTEGER;
 9.                     END;                         ;**NEW**
10.
11.  VAR       P,TOP    :PTR;
12.            INTEXT   :TEXT;
13.            I        :INTEGER;
```

Again, SLP has numbered the lines, and this time it skips two lines after the first occurrence of the string WORDNODE, before inserting the new input line.

You can include update text for several input files in one command file. Type a double slash (//) on a line by itself at the end of the update text for each file. Begin the update text for the next file with a line containing only the command line that specifies the input file to be updated by the next text. Then type the update text on the lines that follow the command line. Type a slash (/) at the end of the command file.

For example, the command file MTST.MAC contains update text to patch the files NUMBER.PAS and GTMSG.MAC, in that order.

```
-/WORDNODE=/+2
               ID  :INTEGER;
//
DY0:NEWMSG=GTMSG.MAC,MTST.MAC
-2,2
    .IDENT   /01.01/
-7
    ADD      A,B
-14
B:  .WORD    0
/
```

Deleting Lines in a File

The SLP command-file command syntax for deleting lines from a file is:

–locator1,locator2,[/audittrail/][;]

where:

locator1	specifies the line where SLP is to begin deleting lines.
locator2	specifies the last line SLP is to delete.

Both locators can be any of the locator forms described in the section, "Formatting the Update Line in a SLP Command File."

If you want to delete lines five through eight in file NUMBER.PAS, it is helpful to look at a numbered listing of NUMBER.PAS.

```
SLP V05.05

 1.  PROGRAM NUMBER;
 2.
 3.  TYPE      TEXT       =FILE OF CHAR;
 4.            PTR        =^WORDNODE;
 5.            WORDNODE=RECORD
 6.                         WORD:ARRAY[1..30] OF CHAR;
 7.                         NEXT:PTR;
 8.                         END;
 9.
10.  VAR       P,TOP      :PTR;
11.            INTEXT     :TEXT;
12.            I          :INTEGER;
```

In the command file, GAMMA.MAC, the command for deleting lines five through eight follows:

```
-/WORDNODE=/,/END/
/
```

When SLP processes GAMMA.MAC with NUMBER.PAS, it produces the following listing file of NUMBER.PAS.

```
*NUMBER.PAS,NUMBER=NUMBER.PAS,GAMMA.MAC RET

SLP V05.05

1.  PROGRAM NUMBER;
2.
3.  TYPE      TEXT      =FILE OF CHAR;
4.            PTR       =^WORDNODE;
5.                                                    ;**-4
6.  VAR       P,TOP     :PTR;
7.            INTEXT    :TEXT;
8.            I         :INTEGER;
```

Replacing Lines in a File

When you replace lines, you delete and then add new text. To replace lines in a file, first enter the full SLP edit command for the delete operation. The first line locator specifies the first line to be deleted. The second line locator specifies both the last line to be deleted and the location where SLP is to insert new text. For example, the following command-file command instructs SLP to move the line pointer to line 4.

```
-4,.+4
```

Then, SLP is to delete the next four lines (represented by +4), including line 4. Finally, SLP is to insert input lines that follow in the command file. SLP inserts the new lines, beginning at the line pointer's current location.

The following examples illustrate replacing lines in a file. First, the source file, BETA.MAC, consists of the following lines:

```
         .TITLE    BETA.MAC
         .MCALL    .TTYOUT, .PRINT, .EXIT
START:   .PRINT    #MESSAG
         MOV       #5,R0
```

Next, the command file, DELTA.MAC, contains:

```
-6,6,/;AUDIT TRAIL/
BNE         START
MOVB (R2),-(R3)
/
```

Then, when SLP processes DELTA.MAC with BETA.MAC, it produces the following listing file:

```
*BETA,BETA=BETA,DELTA.MAC RET
```

```
SLP V05.05

1.            .TITLE   BETA.MAC
2.
3.            .MCALL   .TTYOUT, .PRINT, .EXIT
4.
5.  START:    .PRINT   #MESSAG
6.            BNE      START                        ;AUDIT TRAIL
7.            MOVB     (R2),-(R3)                   ;AUDIT TRAIL
```

Chapter 26

Split File Utility (SPLIT)

The Split File Utility (SPLIT) is an unsupported utility that divides a file along the block boundaries you specify and copies each segment to a separate file. SPLIT is primarily intended for dividing HELP.SAV into its component parts HELP.TXT and HELP.EXE, and for producing SYSMAC.MAC from SYSMAC.SML. However, you can use SPLIT to split other files as well.

Command-Line Syntax

To divide a file, type a command with the following syntax:

SPLIT [outfil1,outfil2,outfil3,...]=infil/option

where:

outfil	specifies the files to which the input file divisions are sent.
infil	specifies the file you want to split.
option	specifies one of the options in the following table.

Options

Option	Function
/B:n[:m]	Defines the boundaries along which to divide the input file; n and m represent the block number (octal) of the beginning of each division, starting with the second. (The beginning of the first division is block 0.) Therefore, you specify one less boundary than the number of divisions you want. For example, to divide a file into three parts you must specify two boundary block numbers.
/H	Displays help information. Use this option by itself, without specifying any input or output files.
/2	Divides the input file in half.

NOTE
SPLIT is not accessible through DCL Commands.

Examples

1. This command divides HELP.SAV into its component parts, HELP.EXE and HELP.TXT:

   ```
   .SPLIT HELP.EXE,,HELP.TXT=HELP.SAV/B:7:12 RET
   ```

   SPLIT writes blocks 0–6 (octal) of HELP.SAV to the file HELP.EXE, discards blocks 7–11 (no second file specification is given in the command line) and writes from block 12 (octal) to the end of HELP.SAV to the file HELP.TXT.

2. This command writes from block 4 to the end of the file SYSMAC.SML to the file SYSMAC.MAC:

   ```
   .SPLIT ,SYSMAC.MAC=SYSMAC.SML/B:4 RET
   ```

   NOTE

   The values used for n and m in the two preceding examples (splitting HELP.SAV and SYSMAC.SML) can change. Refer to CUSTOM.TXT on your distribution kit for the current boundary values. In CUSTOM.TXT, the boundary value variables for splitting HELP.SAV are ..HLP1 and ..HLP2. The boundary value variable for splitting SYSMAC.SML is ..SYSM.

3. This command divides the file BOTH.SAV in half on a block boundary and sends the resulting sections to files ONE.SAV and TWO.SAV:

   ```
   .SPLIT ONE.SAV,TWO.SAV=BOTH.SAV/2 RET
   ```

4. This command requests SPLIT help information:

   ```
   .SPLIT /H RET
   ```

   Note that when you split an ASCII file, division along block boundaries is likely to divide the file in midsentence.

Chapter 27
Transparent Spooling Utility (SPOOL)

The Transparent Spooling Utility (SPOOL) automatically intercepts all data directed to the printer or other designated output device, stores it, and then forwards it to the printer or output device. This allows you to use your terminal while you are printing a file.

This chapter explains how to run SPOOL as a foreground, background, or virtual job. See the *Introduction to RT–11* for tutorial information on running SPOOL as a system job.

Once SPOOL is running, its operations are transparent. Anytime you send output to a printer, either explicitly by issuing commands (such as COPY and PRINT) or by using commands and options that send output to the line printer by default (such as COMPILE/LIST), SPOOL accepts the output and sends it to the printer. While SPOOL runs in the foreground, you can continue to work on other jobs in the background.

SPOOL and QUEUE

SPOOL differs from the Queue Package in that you need not send output to SPOOL as a complete file. Instead, SPOOL accepts output as it becomes available ("pipeline" operation). Queue must wait until a complete file is available before it can begin sending output. Therefore, using SPOOL can be considerably faster than using QUEUE.

The PRINT command is affected when both QUEUE and SPOOL are running. KMON assigns precedence to SPOOL for any PRINT command, so take care if you run both QUEUE and SPOOL. PRINT options /PROMPT and /NAME are specific only to QUEUE. If both QUEUE and SPOOL are running, KMON treats those PRINT options as assigned to SPOOL and returns an invalid option error.

SPOOL's Output Device(s)

Although SPOOL is especially useful for spooling files for printing, the output device is not restricted to a printer. You can use SPOOL for sending files to any RT–11 serial non-file-structured device.

SPOOL is distributed with two default output devices, LP and LS, but you can change the default output device by installing the software customization located in Section 2.7.53 of the *RT–11 Installation Guide*.

In RT–11 5.6, SPOOL supports multiple output devices. See the Running SPOOL as a Virtual Job section for more information on running SPOOL with multiple output devices.

SPOOL's Components

SPOOL consists of a package of three components: a utility program, a pseudo-device handler, and a work file:

- Utility (**SPOOL.REL**) or (**SPOOL.SAV**)

  SPOOL gathers output directed to the printer or other output device, stores (*spools*) it in a work file, and sends the output to the printer or other designated output device. SPOOL runs as a foreground job on an unmapped monitor and as a system or virtual job on a mapped monitor.

  The version of SPOOL used with an unmapped monitor is SPOOL.REL, while that used with a mapped monitor is SPOOL.SAV. SPOOL.SAV occupies significantly less low memory than SPOOL.REL.

- Handler (**SP.SYS**) and (**SPX.SYS**)

  SP is a pseudo-device handler for SPOOL. The handler file is SP.SYS for an unmapped multi-job monitor and SPX.SYS for a mapped monitor.

  When output is directed to the printer, the SP pseudo-device handler causes SPOOL to receive the data and spool it in the work file SPOOL.SYS. As soon as one block of information is available in SPOOL.SYS, SPOOL begins sending the output to the printer. Since SPOOL runs as a foreground or system job, you can continue working in the background while files are spooled and printed.

  The spooler handler can be fetched by SPOOL and need not be loaded in memory unless you are directing output to it from a system or foreground job.

  SPX.SYS (the mapped version of the handler) supports up to eight input devices; the handler syntax is SPn, where *n* is 0 to 7.

- Work File (**SPOOL.SYS**)

  SPOOL.SYS is the work file where SPOOL stores output before sending it to the printer or other output device.

  SPOOL attempts to create its work file, SPOOL.SYS, on the device whose logical name is SFD (Spool File Device). If you have not assigned the logical name SFD to a device, SPOOL creates SPOOL.SYS on the system volume.

  SPOOL by default allocates 1000 (decimal) blocks on SFD or SY for its work file SPOOL.SYS. You can change the default size of SPOOL.SYS by applying the software customization located in Section 2.7.52 of the *RT–11 Installation Guide*.

  If there is not enough room on the volume for a work file of the default size, SPOOL.SYS occupies the largest empty area on the volume. Note that you can use VM as the work device. See the Attaching VM to SPOOL section in this chapter for more information.

  Do not squeeze the volume on which SPOOL.SYS resides while spooling is in progress. This can move SPOOL.SYS, causing unpredictable results.

Running SPOOL as a Foreground or System Job

To use SPOOL:

- Be sure the output device's handler is loaded.
- Run SPOOL.
- Assign the SP pseudohandler the name of the output device as a logical name.

The following sections describe the commands you must issue to run SPOOL. You can include the commands in your start-up indirect command files so SPOOL is automatically available whenever you run under an unmapped multi-job monitor or a mapped monitor.

1. **Loading the Output-Device Handler (usually the printer)**

   Use the SHOW command to see if the LP handler is loaded. If it is not, load the LP handler by typing this command:

   ```
   .LOAD LP  RET
   ```

   If you are running on a Professional 300 series computer, or you have a serial-line printer, load the LS handler instead:

   ```
   .LOAD LS  RET
   ```

   If you customize your system to use an output device other than a printer, substitute your output device's mnemonic for LP or LS.

   You need not load the SP handler itself.

2. **Running SPOOL**

   You can run SPOOL as a foreground or system job. If you are running under an unmapped multi-job monitor, you must set the USR to NOSWAP (SET USR NOSWAP) before running SPOOL. After you issue the command to run SPOOL, you can allow the USR to swap (SET USR SWAP). Under a mapped monitor, you need not set the USR to NOSWAP to run SPOOL.

   To run SPOOL as a foreground job, type this command:

   ```
   .FRUN SPOOL.REL/BUF:256.  RET
   ```

   To run SPOOL as a system job, type this command:

   ```
   .SRUN SPOOL.SAV  RET
   ```

   The FRUN command assumes SPOOL.REL and SP.SYS are on the default volume (DK). The SRUN command assumes SPOOL.SAV and SPX.SYS are on the system volume (SY). If SPOOL.REL or SPOOL.SAV is on another volume, include the device mnemonic in the command (ddn:SPOOL).

   NOTE

   The option /BUF:256. should not be included in the command to run SPOOL when running under a mapped monitor. SPOOL will allocate working space in extended memory.

3. **Assigning a Logical Name to SP**

   For SPOOL to work transparently, you must assign the device mnemonic of the line printer as a logical name for SP, the SPOOL pseudohandler. This causes SP to intercept output directed to the line printer.

   To assign LP as the logical device name, type this command:

   ```
   .ASSIGN SP0: LP:
   ```

   To ensure that logical LP: and LP0: are the same, also type this command:

   ```
   .ASSIGN SP0: LP0:
   ```

   If you want SPOOL to intercept output directed to another physical device, assign that device's mnemonic as SP0's logical device name. For example, if you want SPOOL to intercept all output directed to LS, make the following logical assignment:

   ```
   .ASSIGN SP0: LS:
   ```

Starting SPOOL from a Command File

If you want SPOOL to run automatically whenever you run RT–11, include a sequence of commands like the following in your start-up command file. These commands run SPOOL as a foreground job under an unmapped multi-job monitor:

```
FRUN SY:SPOOL/BUF:256./PAUSE
ASSIGN LS: LP:   †
LOAD LP:=F
SET USR NOSWAP
RESUME
ASSIGN SP0 LP
ASSIGN SP0 LP0
SET USR SWAP
```

Note that SPOOL, as part of its startup procedure, locks the USR for approximately 0.5 seconds to ensure that SPOOL can process commands before further commands are allowed.

† Enter this command line only when using a Professional 300 series processor.

Running SPOOL as a Virtual Job

The following discussion of SPOOL applies to only SPOOL.SAV and the SPX spooler handler.

SPOOL accepts concurrent printing input from up to eight jobs and can direct files to up to eight output devices. You can concurrently print output from more than one job on the same printer or multiple printers.

Configuring Your Spooler

If your system is configured with more than one printer, or any printer on your system is controlled by a device handler other than LP or LS, you must configure the spooler for your system. The following directions explain how to configure your spooler. Example spooler start-up command sequences are included in the directions:

1. First in the start-up command sequence, you assign each printer's device handler to a spooler SO device unit (SOn). Because the spooler initializes internal tables based on those assignments, you must make them before you run SPOOL.

   For example, if your system is configured with a parallel-interface printer controlled by LPX.SYS and a serial-interface printer controlled by LSX.SYS, the following commands would be first in the sequence:

   ```
   ASSIGN LP SO0
   ASSIGN LS SO1
   ```

   If another printer is controlled by a device handler other than LP or LS, for example LQ, the command is:

   ```
   ASSIGN LQ SO2
   ```

   The order of assignments between printer handlers and spooler device units is unimportant.

2. After you have assigned all the printer handlers a unique spooler-device unit number, you start the spooler as a system job but delay the command execution to load handlers:

   ```
   SRUN SPOOL.SAV/PAUSE
   ```

3. You then load all the printer device handlers. This example assumes you are using LP, LS, and LQ:

   ```
   LOAD LP=SPOOL
   LOAD LS=SPOOL
   LOAD LQ=SPOOL
   ```

4. Once you have loaded all the handlers, you finish starting the spooler as a system job:

   ```
   RESUME SPOOL
   ```

5. Each unit of the spooler handler (SP) is associated with the corresponding spooler device (SO). SP0 is associated with SO0, SP1 is associated with SO1, and so on. To make the spooler transparent in use, you should associate the appropriate

logical printer name with the spooler handler (SP) and spooler device handler (SO) for the appropriate printer.

For example, RT–11 utilities use the logical printer name LP for printing operations. The PRINT command by default specifies logical LP as the output device. Likewise, requesting an assembly listing on a command line directs the listing output to logical LP.

To complete the association of logical name LP with the parallel-interface printer, logical name LS with the serial-interface printer, and logical-name LQ with the LQ printer:

```
ASSIGN SP0 LP
ASSIGN SP1 LS
ASSIGN SP2 LQ
```

Spooler STRTxx.COM Summary

The set of start-up commands to configure the spooler as previously described are:

```
ASSIGN LP SO0
ASSIGN LS SO1
ASSIGN LQ SO2
SRUN SPOOL.SAV/PAUSE
LOAD LP=SPOOL
LOAD LS=SPOOL
LOAD LQ=SPOOL
RESUME SPOOL
ASSIGN SP0 LP
ASSIGN SP1 LS
ASSIGN SP2 LQ
```

Those commands establish the following connections:

```
Output to "LP" --> SP0 --> SPOOL --> SO0 --> LPX Printer

Output to "LS" --> SP1 --> SPOOL --> SO1 --> LSX Printer

Output to "LQ" --> SP2 --> SPOOL --> SO2 --> LQX Printer
```

Once these start-up commands are run during a reboot, RT–11 utilities by default send output to LP, which is then directed to the parallel-interface printer. Any printer operations you want to direct to the serial-interface printer are sent to LS, and any to the LQ printer are sent to LQ. For example, the following command prints the file *FILNAM.TXT* residing on device *DU2*, on the serial-interface printer:

```
.PRINT/OUTPUT:LS: DU2:FILNAM.TXT RET
```

Sending Output to the Default Printer

You should note that output sent to the default printer name LP does not need to be directed to the LP handler. If you want, for example, a serial-interface printer to be the default printer for the PRINT command, associate the logical LP with the serial interface printer (LS) in the following way:

```
ASSIGN SP1 LP
```

This command presumes that the serial-interface printer handler LS has been assigned the name SO1 and that SO1 is associated with spooler handler unit SP1.

Attaching VM to SPOOL

If you use VM as the work device for SPOOL, you should attach VM to SPOOL. However, that can make VM unavailable for other use. The following series of commands lets you attach VM as the work device for SPOOL but still makes VM available for other use. These commands can be included in your start-up command file; if you include them, be sure to remove or disable any commands you now use to start SPOOL.

Notice in the series of commands that no unit number is coupled with VM for the ASSIGN command. Note also that the commands to run the virtual versions (SAV) of SPOOL as a system job are different from those to run the (REL) version as a system job:

```
ASSIGN LS: SO0:
ASSIGN VM: SFD:
SRUN SPOOL.SAV/PAUSE
LOAD LS:=SPOOL
LOAD VM7:=SPOOL
ASSIGN SP: LP:
ASSIGN SP0: LP0:
RESUME SPOOL
```

VM does not recognize unit numbers. However, specifying a unit number for VM in the preceding LOAD command lets the RT–11 monitor assign more than one function to VM.

SPOOL SET Commands

Although SPOOL operates transparently, you can use the DCL SET command to control spooling operations. See the *RT–11 Commands Manual* for a full description of this command. The format for using the SET command with SPOOL is:

SET SPx condition

where:

x	specifies the unit number of the device.
condition	specifies the condition you want set (banner pages, form feeds, and so on).

You can set several conditions on a single command line by separating the conditions with commas. For example:

```
SET SP0 WIDE,FLAG=3
```

This command sets SPOOL to generate 132-column banner pages, and sets the default number of banner pages to 3.

NOTE

Note that you must unload SP (the SPOOL handler) and load a fresh copy of SP into memory for a SET command to take effect.

The following table lists and explains the SET command conditions (options) for SPOOL. Most of these conditions require you to specify the unit number 0 (SET SP0), because SPOOL as distributed supports only one output device at a time.

Condition	Function
ENDPAGE=n	Appends *n* terminating formfeeds to each file sent to spooled device *x*. If *x* is not specified, ENDPAGE=n appends *n* terminating formfeeds to files sent to all spooled devices.
	SET SPx ENDPAGE=0 suppresses appending terminating formfeeds to each file sent to spooled device *x*. If *x* is not specified, ENDPAGE=0 suppresses appending terminating formfeeds to files sent to all spooled devices.
	You can generate end-page support without flagpage support. To do so, include the new SYSGEN conditional SP$EPS=1 to support end pages without necessarily supporting flagpages, when you generate your version of the operating system.
EXIT	Aborts SPOOL in a synchronous manner. Use this command to abort SPOOL from within a command file so that monitor prompt is not returned until all SPOOL activity is terminated.

Condition	Function
FLAG=n	Sets the number of banner pages (flagpages) to prepend to each file sent to spooled device *x*. If *x* is not specified, causes *n* flagpages to be prepended to all spooled devices. The value *n* can be an integer in the range 0 to 5 (up to 6 flagpages can be specified). The default value for *n* is 0. The command PRINT/FLAGPAGE:n, when a value is specified for n, overrides the SET SP FLAG=n command. When no value is specified for n with the PRINT/FLAGPAGE:n command, the value for n is set by the SET SP FLAG=n command.
FORM0	Issues a form feed on spooled device *n* each time SPOOL encounters block 0 of a file to be printed; this is useful if the output device is part of a multiterminal system, or if the output device handler does not support its own FORM0 option. If *n* is not specified, issues a form feed on all spooled devices each time SPOOL encounters block 0 of a file to be printed. **NOTE** Setting SP0 and either LS or LP to FORM0 simultaneously generates multiple form feeds.
NOFORM0	Turns off FORM0 mode for spooled device *n*. This is the default mode. If *n* is not specified, turns off FORM0 for all spooled devices.
KILL	Aborts output to the *n* spooled device. If *n* is not specified, aborts output to all spooled devices.
NEXT	Stops printing the current file on spooled device *n* and proceeds to next file queued to that device. If *n* is not specified, defaults to device SP0. If no file is queued, the command is ignored.
WAIT	Suspends output to spooled device *x*, while output to any other spooled device continues. If *x* is not specified, suspends output to all spooled devices. SPOOL does not delete information in the work file and SPOOL continues to accept input when SET SP0 WAIT is in effect.
NOWAIT	Resumes sending output to spooled device *x* after output to that device has been suspended. If *x* is not specified, resumes output to all spooled devices.
WIDE	Causes SPOOL to generate 132-column flag pages for specified device *x*. If *x* is not specified, generates 132-column flag pages for all spooled devices.
NOWIDE	Causes SPOOL to generate 80-column flag pages for specified device *x*. If *x* is not specified, generates 80-column flag pages for all spooled devices.

SPOOL Flag Pages and Status

Flag Pages

SPOOL flag page support is included in the distributed monitors. To generate flag pages and specify the number of flag pages, issue the PRINT command with the /FLAGPAGE=n option or the command set SP0 FLAG=n. These commands cause SPOOL to print that number (n) of flag pages for all files subsequently spooled for printing, unless the files are spooled without an associated file name. For example, the command PIP LP:=MYFIL.MAC sends output to the printer without an associated file name, so no flag pages would be generated.

You can exclude SPOOL flag page support through system generation. Excluding SPOOL flag page support saves 927(decimal) words in the monitor.

Status

You can check the spooler's status by issuing the SHOW QUEUE command. This command requires that the file RESORC.SAV is on device SY.

The SHOW QUEUE command tells whether or not each SPOOL device unit and associated output device is active or inactive, the number of blocks spooled for output, and the number of free blocks in SPOOL's work file.

The following example SHOW QUEUE display shows two output devices, LS0 and LP0, attached to SPOOL, with LS0 printing and LP0 idle:

```
.SHOW QUEUE RET
 Unit  Device  Status
 SP0:  LS0:    ACTIVE,      56 blocks spooled
 SP1:  LP0:    IDLE
   944 Free blocks in workfile
```

If QUEUE is running, the SHOW QUEUE command displays a QUEUE status report as well.

Chapter 28
Source Comparison Utility (SRCCOM)

The RT–11 Source Comparison Utility (SRCCOM) compares two text files and lists the differences between them. It is called *source* comparison since it compares ASCII files, the files programmers write as source code for their programs. See Chapter 2 in the *RT–11 System Utilities Manual*, Part I for a description of the BINCOM utility, which compares files in binary code.

SRCCOM can either display the differences listing on a terminal or printer, or store the listing in a file. If there are no differences, SRCCOM displays a message on the terminal indicating that.

Uses

- SRCCOM is useful when you want to compare two similar versions of a text file. The SRCCOM listing highlights the changes made to the file during an editing session. Depending on the options you choose, SRCCOM can list differences in line feeds, tabs, spaces, blank lines, and even differences between upper and lowercase characters. The only character SRCCOM ignores is the RETURN character.

  Note, however, that you must end the last line of a file with a line feed, if you want SRCCOM to notice that line.
- SRCCOM is also useful for creating a command file you can run with the source language patch program (SLP), described in Chapter 25. When you use SRCCOM for creating a command file, you can patch one version of a source file so that it matches another version. The Creating a SLP Command File section describes how to create a command file for SLP.

Calling and Terminating SRCCOM

To call SRCCOM from the system device, respond to the dot (.) displayed by the keyboard monitor by typing:

```
.R SRCCOM  RET
```

The Command String Interpreter (CSI) displays an asterisk (*) prompt at the left margin of the terminal and waits for you to enter a command string. If you respond to the asterisk by only pressing RETURN, SRCCOM displays its current version number and again prompts you for input.

You can type CTRL/C to halt SRCCOM and return control to the monitor when SRCCOM is waiting for input from the terminal. You must type CTRL/C twice to abort SRCCOM at any other time. To restart SRCCOM, type R SRCCOM or REENTER and press RETURN in response to the monitor's dot.

Command-Line Syntax

The syntax of the SRCCOM command line is:

[[outspec][,SLP-spec]=] file-1,file-2[/option...]

where:

outspec	specifies a file or device (for example, a printer) for the listing of differences.
SLP-spec	specifies a file or device for the command file to be run with SLP. See the Creating a SLP Command File section for more information on creating a command file for SLP.
file-1	specifies the first file to be compared.
file-2	specifies the second file to be compared.
option	is one of the options listed in Table 28–2.

Note that:

- You can specify the input files in any order if you want only a comparison. However, if you are creating a patch for use with the SLP utility, then specify the old file first and the new file second.
- You can omit either or both output file specifications. However the output file specifications are position dependent. So, if you specify a SLP file but no output file, you must place a comma before the SLP file specification to denote the absence of an output file.

Table 28–1 lists the command-line defaults.

Table 28–1: SRCCOM Command-Line Defaults

Default Device or File Type	Description
terminal	output device
DK	input device
MAC	file type for input files
DIF	file type for a differences file
SLP	file type for a SLP command file

Text-File Comparisons

SRCCOM examines the two text files line by line, looking for groups of lines that match. When SRCCOM finds a mismatch, it lists the lines from each file that are different. SRCCOM continues to list the differences until a specific number of lines from the first file matches the second file. The specific number of lines that constitutes a match is a variable that you can set with the /L:n option.

If you compare two files that are identical, RT–11 does not create a listing, but displays the following message on the terminal screen:

```
?SRCCOM-I-No differences found
```

If you compare two files that are different, RT–11 produces a listing of the differences and displays the following message on the terminal screen:

```
?SRCCOM-W-Files are different
```

Regardless of the output specification, the differences message always displays on the terminal, indicating whether the files are alike or different.

Using Wildcards with SRCCOM

You can use wildcards to do multiple text-file comparisons. However, you cannot use wildcards when creating a command file to run with SLP.

You can use wildcards in either input file specification (file-1 or file-2) or both. However, a different type of comparison is done depending on whether you use wildcards in only one or in both of the input specifications:

- If you use wildcards in only one of the input specifications, SRCCOM compares the file you specify without wildcards to all variations of the file you specify with wildcards. The wildcard represents that part of the file specification to be varied. Use this method to compare one file to several other files.

  For example, the following command tells SRCCOM to compare the file TEST1.MAC on device DU0 to all files on device DU1 having the name TEST2:

  ```
  *TEST=DY0:TEST1.MAC,DY1:TEST2.*  RET
  ```

  By specifying an output file, you can send the results of all the comparisons to a file rather than to the terminal. In the preceding example, all differences from the comparisons are sent to the file TEST.DIF on device DK.

- If you use wildcards in both input file specifications, the wildcards represent that part of the file specifications that you want to be the same in both files being compared.

  You can use this method to compare several pairs of files; each input file is compared to only one other input file at a time. For example, the following command requests SRCCOM to compare pairs of files; the first input file in each pair has the file name PROG1, and the second has the file name PROG2. The file type of both files in each pair must match:

  ```
  *DU0:PROG1.*,DU1:PROG2.*  RET
  ```

  SRCCOM searches for the first file on DU0 with the file name PROG1, and takes note of its file type. Then, SRCCOM searches DU1 for a file with the file name PROG2 and the same file type as PROG1. If a match is found, SRCCOM compares the two files and lists the differences on the terminal (or sends the differences to an output file, if one is specified). SRCCOM then searches DU0 for more files with the file name PROG1 and DU1 for PROG2 files with matching file types.

SRCCOM Options

Table 28–2 summarizes the operations you can perform with SRCCOM options. You can place these options anywhere in the command string, but it is conventional to place them at the end of the command line.

Table 28–2: SRCCOM Options

Option	Function
/A	Lets you specify an audit trail (a string of characters that marks each updated line of a patched source file). Use /A with the SLP output file specification to create a file that can be used as command file input for the source language patch program SLP (see Chapter 25).
/B	Compares blank lines; normally, SRCCOM ignores blank lines.
/C	Ignores comments (all text on a line preceded by a semicolon) and spacing (spaces and tabs). A line consisting entirely of a comment is still included in the line count.
/D	Creates a listing of the second file specified in the command line containing the differences with the first file marked by vertical bars (|), and the deletions marked by bullets (•).
/F	Includes form feeds in the output listing; SRCCOM normally compares form feeds, but does not include them in the differences listing.
/L[:n]	Specifies the number of lines that determines a match; n is a decimal integer in the range 1 through 311. The default value for n is 3.
/S	Ignores spaces and tabs.
/T	Compares blanks and tabs that appear at the end of a line. Normally SRCCOM ignores these trailing blanks and tabs.
/V:i:d	Used with /D to specify the characters you want SRCCOM to use in place of vertical bars and bullets. This option is useful if your terminal does not display the vertical bar character. Both i and d represent the numeric codes for ASCII characters in the range 40 through 176 (octal), where i represents the code for the insertion character and d the deletion character code.

Interpreting a Differences Listing

To understand how to interpret the output listing, first look at the following two sample FORTRAN text files EXAMP1.FOR and EXAMP2.FOR. Then look at the listing of the differences between them.

Two Example FORTRAN Files

Note the two differences between the two files:

- In line 7, the first file has *go to 10*, while the second file has *go to 100*.
- In line 14, the first file has the variable *radamg* while the second file has the variable *radang*.

Example 1: FORTRAN File with Errors (EXAMP1.FOR)

```
        real function ASIND( x)
        real x
c
c       This FORTRAN callable function returns the ARCSINE
c       of  a specified value as an angle in degrees.
c
        if (ABS( x) .lt. 1.0) go to 10
        ASIND = x * 90.0
        return
c
c       Use trigonometric identity to calculate ARCSINE of X.
c       Then convert radians to degrees.
c
  100   radamg = ATAN( x / SQRT( 1.0-x**2))
        ASIND = radang * 57.29577951
        return
c
        end
```

Example 2: FORTRAN File without Errors (EXAMP2.FOR)

```
        real function ASIND( x)
        real x
c
c       This FORTRAN callable function returns the ARCSINE
c       of  a specified value as an angle in degrees.
c
        if (ABS( x) .lt. 1.0) go to 100
        ASIND = x * 90.0
        return
c
c       Use trigonometric identity to calculate ARCSINE of X.
c       Then convert radians to degrees.
c
  100   radang = ATAN( x / SQRT( 1.0-x**2))
        ASIND = radang * 57.29577951
        return
c
        end
```

A Differences Listing for the Example Files

Note the following command and the listing file EXAMP.DIF produced by that command:

```
*EXAMP=EXAMP.ONE,EXAMP.TWO  RET
*TYPE EXAMP.DIF  RET
```

```
1) DK:EXAMP.ONE
2) DK:EXAMP.TWO
**********
1)1             if (ABS( x) .lt. 1.0) go to 10
1)              ASIND = x * 90.0
****
2)1             if (ABS( x) .lt. 1.0) go to 100
2)              ASIND = x * 90.0
**********
1)1       100   radamg = ATAN( x / SQRT( 1.0-x**2))
1)              ASIND = radang * 57.29577951
****
2)1       100   radang = ATAN( x / SQRT( 1.0-x**2))
2)              ASIND = radang * 57.29577951
**********
```

The first two lines of the listing identify the two files being compared. Each file name and the device on which each file resides are listed; for example:

```
1) DK:EXAMP.ONE
2) DK:EXAMP.TWO
```

The numbers at the left margin have the form *n)m*, where *n* specifies the files compared (either 1 or 2) and *m* specifies the page (indicated by a form feed) of that file on which the specific line is located. In this case, both files have only one page of output.

RT–11 displays 10 asterisks both before and after a section showing one or more differences between two files. In addition, within each section, a line of 4 asterisks separates the two files being compared, thus dividing each difference section into two subsections; for example:

```
**********
1)1             if (ABS( x) .lt. 1.0) go to 10
1)              ASIND = x * 90.0
****
2)1             if (ABS( x) .lt. 1.0) go to 100
2)              ASIND = x * 90.0
**********
```

Each difference section ends with a matching line, used as a reference to identify the location of the differing lines; for example:

```
ASIND = x * 90.0
```

and

```
ASIND = radang * 57.29577951
```

Using the MATCH Option (/L:n)

A match is the number of lines in each file compared that are exactly the same. SRCCOM looks for a match to conclude a differences section. By default three identical lines constitute a match. However, whenever a match is found, SRCCOM lists only the first line of the match in a differences section.

You can set the number of lines that must match by using the /L:n option. This is a decimal number from 1 to 311. For example, the following command produces the following listing for the preceding example files. Note that, unlike the first differences example, there is only one differences section, since a match (seven identical lines) could not be found before the next difference. Note also, that in this case, both files are on the default storage device (DK):

```
*EXAMP1.FOR,EXAMP2.FOR/L:7  RET

1) DK:ASIND.RNO
2) DK:ASIND2.RNO
**********
1)1            if (ABS( x) .lt. 1.0) go to 10
1)             ASIND = x * 90.0
1)             return
1)      c
1)      c      Use trigonometric identity to calculate ARCSINE of X.
1)      c      Then convert radians to degrees.
1)      c
1)        100  radamg = ATAN( x / SQRT( 1.0-x**2))
1)             ASIND = radang * 57.29577951
****
2)1            if (ABS( x) .lt. 1.0) go to 100
2)             ASIND = x * 90.0
2)             return
2)      c
2)      c      Use trigonometric identity to calculate ARCSINE of X.
2)      c      Then convert radians to degrees.
2)      c
2)        100  radang = ATAN( x / SQRT( 1.0-x**2))
2)             ASIND = radang * 57.29577951
**********
```

Using the CHANGEBAR Option (/D[/V:i:d])

When you use the /D option in the SRCCOM command line, SRCCOM creates a listing in which it inserts vertical bars (|) and bullets (•) to denote the differences between the two files in the command line. The vertical bar indicates insertion; the bullet indicates deletion. If you do not specify an output file, SRCCOM displays the listing at the terminal.

If you include the /V:i:d option with /D (you cannot use /V:i:d without /D), you can specify what characters you would like in place of the vertical bar or bullet. The argument i represents the ASCII code (between 40 and 176 octal) for the character you want in place of the vertical bar. The argument d represents the ASCII code (between 40 and 176 octal) for the character you want to use in place of the bullet.

The following command requests SRCCOM to compare EXAMP1.FOR with EXAMP2.FOR:

```
*EXAMP1.FOR,EXAMP2.FOR/D/L:1 RET
```

When SRCCOM processes the preceding command, it displays on the terminal the following listing of EXAMP2.FOR:

```
                real function ASIND( x)
                real x
        c
        c       This FORTRAN callable function returns the ARCSINE
        c       of  a specified value as an angle in degrees.
        c
    |           if (ABS( x) .lt. 1.0) go to 100
                ASIND = x * 90.0
                return
        c
        c       Use trigonometric identity to calculate ARCSINE of X.
        c       Then convert radians to degrees.
        c
    |     100   radang = ATAN( x / SQRT( 1.0-x**2))
                ASIND = radang * 57.29577951
                return
        c
                end
?SRCCOM-W-Files are different
```

Creating a SLP Command File

You can use SRCCOM to create an input command file to SLP (the source language patch program) described in Chapter 25. To do so, specify a SLP file in the indicated position (see the Command-Line Syntax section) of the SRCCOM command line. For the SLP file specification, all you need specify is the file name. By default, SRCCOM gives your SLP file a SLP file type.

Note that if you specify:

- both an output filespec and a SLP filespec, SRCCOM creates both a differences listing and a SLP input command file.
- only an output filespec, SRCCOM creates only a differences listing.
- only a SLP filespec, SRCCOM creates only a SLP input command file.

In the following sample command line, SRCCOM creates the input command file, MODIFY.SLP. This file contains the necessary commands that, when used with SLP, can modify EXAMP1.FOR so that it matches EXAMP2.FOR:

```
*,MODIFY=EXAMP1.FOR,EXAMP2.FOR/A   RET
```

Notice the /A (audit) option in the command line. When you use the /A option, SRCCOM prompts you for the audit trail:

```
Audit trail?
```

The audit trail is a string of characters that keeps track of the update status of each line in the patched source file. For example, by using the /A option you can record in the SLP file your initials and the date on lines of code that change. SLP appends the audit trail as a comment on the right margin of each updated line in the patched source file.

Respond to the audit prompt with a string of up to 11 characters. Do not use a slash (/); SLP uses slashes to surround and identify each audit trail.

The example below shows the contents of the file MODIFY.SLP produced by the preceding example command. Each audit trail (or *update comment*) begins with the semicolon (; —the MACRO–11 comment character). In this case, RH is the programmer's initials and 2-20-88 is the date the programmer made the change. The numbers preceding the comments are the line numbers in the file that will change. Here there is a one-for-one replacement of an old line 7 with a new line 7 and of an old line 14 with a new line 14. The code beneath each comment is the code that will replace the old code:

```
-7,7,/;RH-2*20*88/
        if (ABS( x) .lt. 1.0) go to 100
-14,14,/;RH-2*20*88/
        100   radang = ATAN( x / SQRT( 1.0-x**2))
/
```

When you use the preceding command file with the SLP utility to modify EXAMP1.FOR, you will get the following new version of that FORTRAN file. Note the audit trail (as listed above) is in the right-hand column of the lines that have been changed. An additional audit-trail line beneath each change indicates the number of lines that changed in each modification. Note also that by default the audit trail begins in column 73. This means that on a terminal or printer having an 80-column wide display, audit trails longer than 8 characters are truncated. However, you can change the start column of the audit trail. See Chapter 25 for more information about the audit trail:

```
        real function ASIND( x)
        real x
c
c       This FORTRAN callable function returns the ARCSINE
c       of  a specified value as an angle in degrees.
c
        if (ABS( x) .lt. 1.0) go to 100                         ;RH-2*20*91
        ASIND = x * 90.0                                        ;**-1
        return
c
c       Use trigonometric identity to calculate ARCSINE of X.
c       Then convert radians to degrees.
c
        100   radang = ATAN( x / SQRT( 1.0-x**2))               ;RH-2*20*91
        ASIND = radang * 57.29577951                            ;**-1
        return
c
        end
```

DCL Equivalents of SRCCOM Utility Operations

Table 28–3 lists the DCL DIFFERENCES command options that are equivalent to SRCCOM utility operations.

The first part of the table lists that part of the SRCCOM command syntax that is equivalent to a DIFFERENCES option. The rest of the table alphabetically lists all the SRCCOM options having DCL equivalents. Those SRCCOM options not having DCL equivalents are not listed.

Table 28–3: DCL Equivalents of SRCCOM Utility Operations

SRCCOM Syntax/Option	DIFFERENCES Option
LP:=	/PRINTER
outspec=	/OUTPUT:outspec
[size]=	/ALLOCATE:size
,SLP-filespec=	/SLP:filespec
TT:= (default)	/TERMINAL (default)
/A	/AUDITTRAIL
/B	/BLANKLINES
/D	/CHANGEBAR
/C	/NOCOMMENTS
(default)	/COMMENTS (default)
/F	/FORMFEED
/L:n	/MATCH[:n]
/S	/NOSPACES
(default)	/SPACES (default)
/T	/NOTRIM
(default)	/TRIM (default)

Chapter 29

Native Transfer Utility (TRANSFER/TRANSF)

The Native Transfer Utility (TRANSFER/TRANSF) is an unsupported program that runs on several host operating systems and can be used to copy files from an RT–11 stand-alone processor to the host processor or from the host to the standalone.

A *host* processor is a computer connected to your RT–11 stand-alone computer by means of the VTCOM utility. TRANSFER is run only from the host processor and supports the following host operating systems:

VMS
RSX–11M
RSX–11M–PLUS
Micro/RSX
RT–11

Name Conventions

- In the rest of this chapter, *RSX* refers to RSX–11M, RSX–11M–PLUS, and Micro/RSX.
- The name *Native Transfer Utility* refers to the utility as installed on a host operating system.
- *TRANSF* is the name of the utility, the file containing the utility, and the command when TRANSF.SAV is installed on an RT–11 host. Unlike RSX and VMS, RT–11 allows only up to six characters in a file name, and the name of the utility (when on an RT–11 host) is kept the same as its RT–11 acronym file name.

  TRANSF is also the name of the RT–11 distribution file containing the Transfer Utility (TRANSF.EXE for a VMS host, TRANSF.TSK for an RSX host, and TRANSF.SAV for an RT–11 host).
- *TRANSFER* is the name of the utility, the file containing the utility, and the command used by the utility, when the appropriate version of TRANSF (TRANSF.TSK or TRANSF.EXE) is installed on an RSX or VMS host.

First-Time Users

First-time users of the Transfer utility should read the communications chapter in the features section of the *Introduction to RT–11*. This chapter describes in detail all that is involved to succesfully communicate between a computer running RT–11 and another computer. The chapter also explains to a new user how to use the Transfer utility.

Three Helpful Features of the Transfer Utility

The Transfer utility:

- Allows you to copy the contents of any file between your local and host computers.
- Error checks while transferring your file(s) to ensure you of an accurate copy.
- Together with VTCOM, dynamically controls the transmission packet size of transferred data to ensure the accuracy of the data at the most reliable transmission speed.

  Transmission errors successively halve packet size while continued transmission success progressively restores packet size. The maximum packet size is 512 bytes and the minimum is 16. For example, a lot of continuous transmission errors could reduce the packet size to 16 bytes, and if the transmission problems stopped, the packet size could go back to 512 bytes.

  The process of packet size reduction and restoration is controlled by the program sending the data. Data transfers from the RT–11 stand-alone processor to a host processor are controlled by VTCOM. Data transfers from a host processor to the RT–11 stand-alone processor are controlled by the Transfer utility.

Transfer Utility Requirements

To use the Transfer utility:

- You must install VTCOM on your local computer, the computer to which your console terminal is attached and which is running RT–11. VTCOM establishes the connection between your RT–11 local computer and the host computer. You can use the Transfer utility only after you have established that connection.

  If you intend to run VTCOM as a system job, you must have the file VTCOM.SAV on your system (SY) device and you must LOAD all handlers for the devices you are going to use with VTCOM. The RT–11 SHOW command displays information about devices.

  If you intend to run VTCOM as a foreground job, then you need the file VTCOM.REL on your system (SY) device. See Chapter 31 for complete information on using VTCOM.

- You must install the proper version of the Transfer utility on your host processor. Note that the Transfer utility is run only from the host processor.

  The Transfer utility is called the "*Native* File Transfer Utility" (or from the point of view of the local RT–11 system, the "*Remote* File Transfer Utility") since it is designed to be native to (or a part of) the *host* operating system. So, there is a unique version of the Transfer utility for each type of host operating system on which it can run: RSX, VMS, and RT–11. Each version is a different program with different code adapted to a particular operating system. You can identify the versions by the file *TYPE* of the Transfer utility file. The files containing the different versions of the Transfer utility with the operating systems on which they run are as follows:

Version	Host Operating System
TRANSF.TSK	RSX–11
TRANSF.EXE	VMS
TRANSF.SAV	RT–11

  If your host computer is a PDP–11 running RSX–11, have the host computer system manager install TRANSF.TSK. The *Installing TRANSFER on RSX* section explains how to do this.

  If your host computer is a VAX running VMS, have the host computer system manager install TRANSF.EXE. The *Installing TRANSFER on VMS* explains how to do this.

  If your host computer is a PDP–11 running RT–11, make sure TRANSF.SAV is located on your host system (SY) device.

Installing TRANSFER

The Transfer utility as installed on VMS or RSX is called TRANSFER. So, most of this chapter (the following sections dealing the VMS and RSX) refers to the utility by that name.

TRANSFER can copy files from Files–11 volumes (on VMS or RSX) to RT–11 volumes or the reverse (from RT–11 volumes to Files–11 volumes). Files–11 is the disk volume format maintained by the VMS and RSX operating systems.

You can use command qualifiers to specify the format of the transferred output file. The supported formats are:

- ASCII
- Binary
- FORTRAN
- Image

In addition, you can use command qualifiers that cause TRANSFER to:

- Supply additional information during the transfer operation
- Display HELP information
- Operate in PROMPT mode, an interactive mode with questions and defaults
- Queue the output file to the host system default printer (VMS host only)

Installing TRANSFER on the Host

Before you can use TRANSFER, you must install the appropriate version on the host system. The program named TRANSF.EXE is the version that runs on VMS, and the program named TRANSF.TSK is the version that runs on RSX.

Two Ways of Installing TRANSFER

TRANSFER can be installed on the host processor in two ways. It can be installed as a system utility that is available to all system users or it can be installed in an individual's account.

- The host system manager must install TRANSFER if it is to be a system-wide utility.
- You can install TRANSFER in your own account on the host. If you want to do so, follow the procedures in the section *Installing TRANSFER on VMS* or in the section *Installing TRANSFER on RSX*. However, you must have a privileged account to install TRANSFER on an RSX host. If you do not have a privileged account on an RSX host, you must have the system manager install TRANSFER in your account. You do not need a privileged account to install TRANSFER on a VMS host.

Installing TRANSFER on VMS

Install TRANSFER on Version 4.0 or subsequent versions of VMS.

Step 1: Copy TRANSF.EXE

Mount the RT–11 distribution volume on the host and issue the following VMS command to transfer TRANSF.EXE to the host. When you type the command, replace the variable *ddcu:* with the VMS physical device name for the device on which TRANSF.EXE resides. Type the command exactly as shown:

```
$ EXCHANGE COPY/CONTIGUOUS ddcu:TRANSF.EXE/VOLUME_FORMAT=RT11 TRANSFER.EXE
```

NOTE

Because TRANSF.EXE is not an ASCII file, you cannot use the VTCOM SEND command to copy TRANSF.EXE from the RT–11 stand-alone processor to the host.

Step 2: Define TRANSFER as a Foreign Command

The following command completes the installation of TRANSFER by defining it as a foreign command. (A foreign command is a symbol name, in this case TRANSFER, that you use to invoke the utility.) Placement of the asterisk (*) lets you abbreviate TRANSFER to TRA.

When you type the following command, replace the variable *disk* with the name of the disk on which your VMS directory resides. Be sure to include the dollar sign ($) before the variable *disk*. Replace the variable *directory* with the name of your VMS directory:

```
$ TRA*NSFER :== $disk:[directory]TRANSFER.EXE
```

Installing TRANSFER on RSX

Install TRANSFER on the indicated version or subsequent versions of the operating systems in the following list. The RMSRES resident library must be installed on any RSX operating system running TRANSFER. The DAPRES resident library must be installed, along with system DECnet support, to access files across DECnet.

- RSX–11M–PLUS 3.0
- RSX–11M 4.2
- Micro/RSX 3.0

Step 1: Copy TRANSF.TSK

Mount the RT–11 distribution volume on the host and use the following RSX commands to transfer TRANSF.TSK to the host. When you type the commands, replace the variable *ddn:* with the RSX physical device name for the device on which TRANSF.TSK resides:

```
>MOUNT/NOSHARE/FOREIGN ddn:
```

```
>FLX SY:=ddn:TRANSF.TSK/CO/RT/IM
```

```
>RENAME TRANSF.TSK TRANSFER.TSK
```

NOTE

Because TRANSF.TSK is not an ASCII file, you cannot use the VTCOM SEND command to transfer TRANSF.TSK from the RT–11 stand-alone processor to the host.

Step 2: Install the RSX Task Image

Use the following command to install TRANSFER. You must have a privileged account to issue this command. If you do not have a privileged account, have your system manager install TRANSFER:

```
>INSTALL TRANSFER.TSK
```

Using TRANSFER

With TRANSFER, you can copy files between a PDP–11 processor running RT–11 and either a PDP–11 processor running RSX or a VAX processor running VMS. No intermediary (such as the RT–11 emulator, RTEM) is required.

Requirements

To use TRANSFER, make sure the appropriate version of the TRANSFER utility has been installed on the host processor (running VMS or RSX). Then use VTCOM to establish a connection between the RT–11 standalone processor and the host. You can use the TRANSFER utility only after you have established that connection.

If you run VTCOM as the foreground or system job, you must LOAD all the handlers for the devices you are using with VTCOM. The RT–11 SHOW command displays information about devices.

Issuing the TRANSFER Command

Run the TRANSFER utility by issuing the TRANSFER command in response to the host system's prompt.

Aborting TRANSFER

You can abort a file transfer at any time and return to the host system's prompt by typing CTRL/C. CTRL/C is the preferred method of aborting file transfers on RSX and VMS systems. You can also abort file transfers on VMS systems by typing CTRL/Y and then issuing the EXIT command in response to the monitor prompt.

You can exit from TRANSFER and return to the host system's prompt by typing CTRL/Z in response to a TRANSFER prompt.

TRANSFER Command-Line Syntax

TRANSFER infile/qualifier[s] [outfile/qualifier[s]]

where:

infile Specifies the file you want to copy. The file specification is operating system dependent. Consult your operating system documentation for the correct construction of the file specification. However, observe the following:

- Wildcards are not allowed in the file specification.
- If you do not specify an output file, the first six characters of the input file name and the first three characters of the input file type (extension) must be alphanumeric.
- TRANSFER does not access files contained within a virtual disk on the host.
- You cannot specify any host system record-oriented device, such as a line printer, terminal, or magtape device.

outfile Specifies the file to which you want to transfer the input file. The file specification is operating system dependent. Consult your operating system documentation for the correct construction of the file specification. However, observe the following:

- Wildcards are not allowed in the file specification.
- If you do not specify an output file, that file will have the same name as the input file, with one exception: RT–11 truncates input file names to six characters and input file types (extensions) to three characters.
- You cannot specify any host system record oriented device, such as a line printer, terminal, or magtape device.

/qualifier Specifies a TRANSFER qualifier. There are two types of TRANSFER qualifiers. Mode qualifiers determine the format of the transferred file. Control qualifiers affect transfer file processing by providing information, by invoking interactive mode, or by queuing the output file to a printer. The following table lists all the qualifiers.

Command Qualifiers	File Qualifiers (input or output)
{ /HELP /PROMPT /VERSION }	{ /ASCII[:n] /BINARY[:n] /FORTRAN[:n] /IMAGE[:n] }
	/LOG
	/PROGRESS[:n]
	{ /REMOTE /TERMINAL }
	/STATISTICS

{ } Specifies mutually exclusive qualifiers, which are mutually exclusive on the input or output file specifications and must appear only once in the command.

Default File Types

TRANSFER recognizes certain file types. If you do not specify a mode qualifier, TRANSFER performs the transfer in the default mode for that file type. TRANSFER recognizes the following file types as being ASCII or Binary.

ASCII (variable-length records)

```
.ANS  .BAK  .BAS  .BAT  .BLI  .B16  .B2S  .B32  .C
.CBL  .CMD  .COM  .COR  .CTL  .DAT  .DBL  .DDF  .DIF
.DIR  .DMP  .DOC  .FOR  .FTN  .H    .LIS  .LOG  .LST
.MAC  .MAP  .MAR  .MEM  .ODL  .PAS  .REQ  .RNO  .R16
.R32  .S    .SLP  .SRC  .TEC  .TES  .TXT
```

Binary (variable-length records)

```
.BIN  .LDA  .OBJ  .STB
```

IMAGE mode is TRANSFER's default transfer mode. If you do not specify the /ASCII, /BINARY, /FORTRAN, or /IMAGE qualifier, and if the file you specify does not have a recognized file type, TRANSFER performs the transfer operation in IMAGE mode with 512-byte fixed-length records.

Summary of TRANSFER Qualifiers

The TRANSFER command has two types of qualifiers (options): Mode qualifiers that determine the format of a transferred file, and control qualifiers that determine how the file is processed.

Format conversions can be in either direction between Files–11 volumes and RT–11 volumes. However, you can specify only one TRANSFER mode qualifier in a command.

Table 29–1 summarizes the qualifiers.

Table 29–1: Summary of TRANSFER Qualifiers

Qualifier	Mode	Function
/ASCII[:n]	Mode	Formats the output as ASCII.
/BINARY[:n]	Mode	Formats the output as a binary OBJ file.
/FORTRAN[:n]	Mode	Transfers files containing FORTRAN carriage-control characters.
/HELP	Control	Displays information about TRANSFER.
/IMAGE[:n]	Mode	Transfers files without performing any record translations on them.
/LOG	Control	Creates a log of the names of all the files transferred.
/PROGRESS[:n]	Control	Displays the progress of the transfer at specific intervals while the operation is taking place.
/PROMPT	Control	Operates in an interactive mode that displays questions, indicates defaults, and accepts input.
/REMOTE	Control	Specifies the file on the local RT–11 system.
/TERMINAL	Control	Specifies the file on the local RT–11 system (this qualifier is identical to /REMOTE).
/SPOOL	Control	Queues the TRANSFER output file to the default VMS printer queue.
/STATISTICS	Control	Displays the number of retries and the number of characters saved through compression encoding.
/VERSION	Control	Displays the TRANSFER utility's version number.

TRANSFER Qualifier Descriptions

/ASCII[:n]

Formats the output file as ASCII. On an RT–11 output volume, the file contains ASCII data records, each terminated by a carriage return/line feed, escape, form feed, or vertical tab. TRANSFER removes rubouts, nulls, and vertical tabs from input records and adds carriage return/line-feed pairs to the end of records that do not end with escape, form feed, or line feed. When the host input file record attributes do not specify carriage control, TRANSFER assumes embedded carriage control. Embedded carriage control means that each record contains the control characters necessary for proper formatting. In that case, carriage return/line feed pairs are not appended to records.

In transfers from RT–11 to Files–11 volumes, TRANSFER removes carriage return/line-feed pairs from the end of records.

If you specify a size value (:n), TRANSFER generates fixed-length records of that size and pads them with nulls if required. If you omit the size value (or specify a size of zero), TRANSFER generates variable-length records.

/BINARY[:n]

Formats the output file as a binary object (.OBJ) file. (Do not use /BINARY with library files; use /IMAGE instead.) TRANSFER adds formatted binary headers and checksums to records that it copies to RT–11 files and removes binary headers and checksums from records that it copies to Files–11 files.

When you transfer files to Files–11 volumes, TRANSFER generates fixed-length records of the size (:n) you specify. If you omit the size value (or specify a size of zero), TRANSFER generates variable-length records.

/FORTRAN[:n]

Transfers files that contain FORTRAN carriage control characters. Use the /FORTRAN qualifier when the first character of each record is to be interpreted as the carriage control specifier. The /FORTRAN qualifier does not alter any record data. Use this qualifier only when the output volume is a Files–11 volume.

TRANSFER generates fixed-length records of the size (:n) you specify. If you omit the size value (or specify a size of zero), TRANSFER generates variable-length records.

/HELP

Displays limited information about TRANSFER. The /HELP qualifier lists default TRANSFER formats for various file types.

/IMAGE[:n]

Transfers files without performing any record translations on them. In other words, TRANSFER copies the files exactly as they are.

When you transfer files to Files–11 volumes, TRANSFER generates fixed-length records of the size (:n) you specify. If you omit the size value (or specify a size of zero), TRANSFER generates fixed-length records of 512 bytes.

/LOG

Creates a log of the names of all files transferred. The log displays a success message, the complete input and output filespecs, and the number of blocks or records transferred. For example:

```
$ TRANSFER FOO.TXT/LOG FOO.TXT
%TRANSFER-S-COPIED, USER:[RTINDEX]FOO.TXT;2 copied to __TTB0::DK:FOO.TXT (6 blocks)
```

/PROGRESS[:n]

Displays the progress of the transfer at specific intervals while the operation is taking place. Progress is displayed in record or block intervals, depending on the format of the file and the direction of the transfer. You can specify the interval (:n). The default PROGRESS report interval is 10(decimal) records or blocks.

The /PROGRESS qualifier displays an informational message, the time-of-day, the blocks or records transferred during that interval, the number of retries, and the packet size. An increase in the number of retries and a decrease in the packet size generally indicate interference on the transmission line. If the packet size decreases to an unacceptable level, retry the transfer operation when your transmission line might be more clear.

An example of the /PROGRESS qualifier:

```
$ TRANSFER FOO.TXT/PROGRESS:1 FOO.TXT
%TRANSFER-I-PROGRESS, 14:33:25 blocks transferred=1 retries=0 packet_size=512
%TRANSFER-I-PROGRESS, 14:33:27 blocks transferred=2 retries=0 packet_size=512
```

/PROMPT

Causes TRANSFER to operate in an interactive mode that displays questions, indicates defaults, and accepts input. The questions and defaults change dynamically, based on the responses to earlier questions. PROMPT mode shows a list of available responses in parentheses. The default response appears in brackets. To choose the default response, press RETURN. Choose the default unless you are sure that another qualifier is correct.

The following are sample PROMPT mode sessions:

1. This example copies the ASCII file V5NOTE.TXT from the RT–11 stand-alone system to the host, keeping the same file name. Defaults are taken for the file format (ASCII) and host file record length (variable):

```
$ TRANSFER/PROMPT
Original file is on (HOST,REMOTE) [HOST]: remoteRET
Name of original REMOTE file to copy: v5note.txtRET
Name of file to create on HOST [V5NOTE.TXT]: RET
Create HOST file with (ASCII,BINARY,FORTRAN,IMAGE) records [ASCII]: RET
Create HOST file with fixed-length records (YES,NO) [NO]: RET
%TRANSFER-S-COPIED, __TTA2::DK:V5NOTE.TXT copied to USER:[RT11]V5NOTE.TXT
19
(714 records)
There were 0 retries, with 3362 characters saved through compression encoding.
```

2. This example copies the file VDT.OBJ from the host to an RT–11 stand-alone system, keeping the same file name. Because VDT.OBJ is a binary file, the default BINARY qualifier is chosen by pressing RETURN:

```
$ TRANSFER/PROMPT
Original file is on (HOST,REMOTE) [HOST]: RET
Name of original HOST file to copy: vdt.objRET
Name of file to create on REMOTE [VDT.OBJ]: RET
Create REMOTE file with (ASCII,BINARY,FORTRAN,IMAGE) records [BINARY]: RET
%TRANSFER-S-COPIED, USER:[WINNING]VDT.OBJ 1 copied to __TTA2::DK:VDT.OBJ
(8 blocks)
There were 0 retries, with 276 characters saved through compression encoding.
```

3. This example copies the image file (executable program) VTCOM.SAV from the RT–11 system to the host, keeping the same file name. All defaults are taken:

```
$ TRANSFER/PROMPT
Original file is on (HOST,REMOTE) [HOST]: remoteRET
Name of original REMOTE file to copy: vtcom.savRET
Name of file to create on HOST [VTCOM.SAV]: RET
Create HOST file with (ASCII,BINARY,FORTRAN,IMAGE) records [IMAGE]: RET
Create HOST file with fixed-length records of size (0-512) <512>: RET
%TRANSFER-S-COPIED, __TTC2:DK:VTCOM.SAV copied to USER:[RT11]VTCOM.SAV 2
(24 blocks)
There were 0 retries, with 5274 characters saved through compression encoding.
```

/REMOTE and /TERMINAL

The /REMOTE and /TERMINAL qualifiers indicate to TRANSFER which file is on the local RT–11 system. /REMOTE and /TERMINAL are interchangeable; they have the same meaning. Do not specify both qualifiers in the same command.

To transfer a file from the host system to the stand-alone RT–11 system, use /REMOTE or /TERMINAL on the output-filespec. That operation is the default, so you can omit those qualifiers altogether from the command for this type of transfer.

To transfer a file from the stand-alone RT–11 system to the host system, use /REMOTE or /TERMINAL on the input-filespec.

The following example transfers the file FOO.TXT from the host to the stand-alone RT–11 system:

```
$ TRANSFER FOO.TXT FOO.TXT
```

The next example transfers the file FOO.TXT from the stand-alone RT–11 system to the host:

```
$ TRANSFER FOO.TXT/TERMINAL FOO.TXT
```

/SPOOL

Queues the TRANSFER output file to the default VMS system printer queue. Specify the /SPOOL qualifier on the host output-filespec.

This qualifier is available only for VMS system transfers.

/STATISTICS

Displays the number of retries and the number of characters saved through compression encoding. (Compression encoding is a transfer speed enhancement used by TRANSFER whether or not you specify the /STATISTICS qualifier.) For example:

```
$ TRANSFER FOO.TXT/STATISTICS FOO.TXT
There were 0 retries, with 400 characters saved through compression
```

/VERSION

Displays the TRANSFER utility's version number.

Message Format of the TRANSFER Utility

The messages displayed by the TRANSFER utility when it is installed on either the RSX or the VMS operating system have the following format:

%ProgranName-ErrorLevel-MessageAbbreviation, MessageText
[-ProgramName-ErrorLevel-MessageAbbreviation, MessageText]

where:

Program Name specifies the TRANSFER utility or operating system facility or component name. A percent sign (%) prefixes the first message issued, and a hyphen (-) prefixes each subsequent message.

Error Level is a single letter indicating the severity of the error. The error level can be one of the following.

Error Level	Meaning
S	Success
I	Information
W	Warning
E	Error
F	Fatal or severe error

Consult your host operating system documentation for the meaning of each severity level.

Message Abbreviation is the abbreviation of the message text; the following messages are alphabetized by this abbreviation.

Message Text is the explanation of the message.

[-ProgramName-ErrorLevel-MessageAbbreviation, MessageText] is the next message.

TRANSFER Message Descriptions

The following is a listing of TRANSFER utility error messages. They are alphabetically arranged according to the message abbreviation. Included is an explanation of each message and the recommended action to be taken. Consult your host operating system documentation for messages not listed here.

AMBIGQUAL, ambiguous qualifier in command

Explanation: The command qualifier abbreviation contains too few characters to make it unique. Another qualifier begins with the same characters.

User action: Retype the command, using at least four characters of the qualifier name.

CHKSUMERR, binary record checksum error

Explanation: The checksum for a BINARY record was in error.

User action: Make sure the input file is not an object library. Object libraries must be copied in IMAGE mode rather than BINARY mode.

Make sure the input file is a valid binary file. Retry the transfer.

CLOSEIN, error closing 'input-file' as input

Explanation: RMS encountered an error while closing the indicated input file. This message is usually accompanied by an RMS message indicating the reason for the failure.

User action: Take corrective action based on the associated message.

CLOSEOUT, error closing 'output-file' as output

Explanation: RMS encountered an error while closing the indicated output file. This message is usually accompanied by an RMS message indicating the reason for the failure.

User action: Take corrective action based on the associated message.

CONFQUAL, conflicting qualifier in command

Explanation: You specified qualifiers that are mutually exclusive. For example, you can specify only one TRANSFER mode qualifier (/ASCII, /BINARY, /FORTRAN, or /IMAGE) in a command. Also, you can specify /REMOTE or /TERMINAL on only one side of a command.

User action: Correct the qualifiers and retry the operation.

FILSYNTAXERR, error in file specification

Explanation: You specified a file containing syntax that was invalid for the RT–11 system. For example, the file name might contain more than six characters.

User action: Correct the syntax of the file specification for the RT–11 system.

ILLBINFORMAT, illegal binary record format

Explanation: You attempted to transfer a file from the RT–11 system to the host system using the /BINARY format qualifier. The file either was not a binary file or was a binary file with a bad format.

User action: Make sure the specified record format qualifier agrees with the actual file record format. Retry the operation.

INVCMDSYNTAX, invalid command syntax

Explanation: The TRANSFER utility command contained invalid syntax.

User action: Correct the syntax and reenter the command.

INVINPUTQUAL, invalid input qualifier /′qualifier′

Explanation: The indicated input qualifier is invalid in the command.

User action: Correct the qualifier and reenter the command.

INVQUAL, invalid qualifier /′qualifier′

Explanation: The indicated qualifier is invalid in the command.

User action: Correct the qualifier and reenter the command.

INVQUALVAL, invalid value for /′qualifier′ qualifier

Explanation: You specified an invalid value for the indicated qualifier.

User action: Check the range of valid values for that qualifier. Correct the value for the qualifier and reenter the command.

NOVTCOM, VTCOM not running on remote

Explanation: VTCOM is not running on your RT–11 stand-alone system, or the host system response time is slow due to heavy usage. TRANSFER timed out before receiving a response from VTCOM.

User action: Make sure VTCOM is running on the RT–11 system. Decrease the load on the host system or lower the baud rate between the RT–11 stand-alone system and the host. Retry the operation.

OPENIN, error opening ′input-file′ as input

Explanation: The indicated input file cannot be opened. This message is usually accompanied by an RMS message indicating the reason for the failure.

User action: Take corrective action based on the associated message.

OPENOUT, error opening ′output-file′ as output

Explanation: The indicated output file cannot be opened. This message is usually accompanied by an RMS message indicating the reason for the failure.

User action: Take corrective action based on the associated message.

QUOTNOOUTPUT, quoted string specification as input needs output specification

Explanation: You did not specify the output (RT–11 only) file, when the input (RMS) file was a quoted string specification.

User action: Explicitly specify the output file when the input file is a quoted string specification.

READERR, error reading 'input-file'

Explanation: The indicated input file cannot be read. This message is usually accompanied by an RMS message indicating the reason for the failure.

User action: Take corrective action based on the associated message.

REMABORT, file transfer aborted by remote

Explanation: The file transfer was aborted due to an I/O error on the RT–11 system, or the VTCOM RESET command was issued.

User action: Check the procedures listed in Section 2 of the *RT–11 System Message Manual* for recovery from hard error conditions.

REMACCESS, error accessing remote

Explanation: A hard error occurred on the host system during a file transfer.

User action: Retry the operation after performing hard error recovery procedures on the host system.

REMCLOSEIN, error closing 'input-file' as input on remote

Explanation: TRANSFER encountered an error while closing the indicated input file on the RT–11 system. This message is usually accompanied by another message indicating the reason for the failure.

User action: Take corrective action based on the associated message.

REMCLOSEOUT, error closing 'output-file' as output on remote

Explanation: TRANSFER encountered an error while closing the indicated output file on the RT–11 system. This message is usually accompanied by another message indicating the reason for the failure.

User action: Take corrective action based on the associated message.

REMOPENIN, error opening 'input-file' as input on remote

Explanation: TRANSFER cannot open the indicated file on the RT–11 system. This message is usually accompanied by another message indicating the reason for the failure.

User action: Take corrective action based on the associated message.

REMOPENOUT, error opening ′output-file′ as output on remote

Explanation: TRANSFER cannot open the indicated output file on the RT–11 system. This message is usually accompanied by another message indicating the reason for the failure.

User action: Take corrective action based on the associated message.

REMREADERR, error reading ′input-file′ on remote

Explanation: TRANSFER cannot read the indicated input file on the RT–11 system. This message is usually accompanied by another message indicating the reason for the failure.

User action: Take corrective action based on the associated message.

REMTIMEOUT, remote timed out during file transfer

Explanation: TRANSFER timed out during a file transfer.

User action: Make sure VTCOM is running. Retry the operation.

REMWRITEERR, error writing ′output-file′ on remote

Explanation: TRANSFER encountered an error while writing the indicated file on the RT–11 system. This message is usually accompanied by another message indicating the reason for the failure.

User action: Take corrective action based on the associated message.

SENSEMODE, couldn′t read terminal characteristics

Explanation: TRANSFER could not read the terminal hardware characteristics required for a file transfer operation.

User action: Retry the operation after performing hard error recovery procedures on the host system.

SETMODE, couldn′t write terminal characteristics

Explanation: TRANSFER could not set the terminal hardware characteristics required for a file transfer operation.

User action: Retry the operation after performing hard error recovery procedures on the host system.

TERMINIT, error initializing terminal for I/O

Explanation: TRANSFER could not initialize the terminal hardware on the host system for a file transfer operation.

User action: Retry the operation after performing hard error recovery procedures on the host system.

TOOBIG, record too large for file′s I/O buffer

Explanation: You attempted to transfer a Files–11 record that was too large for TRANSFER′s file buffer.

User action: Make sure you are transferring a file in the correct mode or transfer the file in IMAGE mode.

TOOLONG, ′nn′ byte record exceeds output file′s maximum record length

Explanation: You specified a record length that was too small to hold the largest record from the RT–11 file.

User action: Specify a record length large enough to hold the largest record from the RT–11 file or transfer the file using variable-length records.

USERABORT, file transfer aborted by user

Explanation: You aborted the file transfer by typing CTRL/C, or an I/O error occurred on the host system.

User action: If an I/O error occurred, perform hard error recovery procedures on the host system. Retry the operation.

WRITEERR, error writing ′output-file′

Explanation: RMS encountered an error while writing the indicated file. This message is usually accompanied by another message indicating the reason for the failure.

User action: Take corrective action based on the associated message.

Installing TRANSF on an RT–11 Host

The Transfer utility, when installed on an RT–11 or RTEM–11 host, has the acronym TRANSF as its name — since RT–11 utilities are traditionally named by the file containing the utility, and RT–11 file names can only be 6 characters in length. Therefore, in the rest of this section, the utility is called TRANSF, and the messages displayed by TRANSF on RT–11 or RTEM–11 are alphabetically listed in the *RT–11 System Message Manual* under the name TRANSF.

RT–11 host computers have the same file structure as your local computer. Therefore, TRANSF does no format translation and has no format options. All file transfers take place in image mode.

TRANSF must be installed on a host running RT–11 or RTEM–11. TRANSF runs only on the host processor. Do not run TRANSF on your local terminal.

To install TRANSF on the host, you must copy TRANSF.SAV from your RT–11 distribution or system volume to a common volume, carry the volume to the host, and copy TRANSF.SAV from the volume to the host.

TRANSF Command-Line Syntax

To run TRANSF on your host system, type a command with the following CCL syntax in response to your host system's prompt:

TRANSF outfile[/options]=infile[/options]

where:

outfile	specifies the file (the device, file name, and file type) to which you want a file copied.
infile	specifies the file (the device, file name, and file type) you want to copy.
options	specify the following options.

Option	Function
/S	Rings terminal bell when log messages are printed during file transfers. Automatically enables log messages.
/T	Specifies which file is the RT–11 stand-alone system file. To copy a file from the host to your stand-alone system, use /T with the output-filespec. To copy to the host, use /T with the input-filespec. If you do not specify /T in the command line, TRANSF assumes you are copying from the host to your stand-alone system. You cannot use /T on both sides of the command string.
/W	Causes TRANSF to print log messages during file transfers, but does not ring the terminal bell.

RT–11 file specifications can include only a device, a file name of up to six characters, and a three-character file type. You cannot use wildcards in any file specifications for TRANSF.

Using TRANSF

Use the preceding command format to transfer files from an RT–11 host to your stand-alone system and from your stand-alone system to an RT–11 host.

NOTE

If the host system supports the XON/XOFF feature, TRANSF can transfer files at any baud rate you choose. However, if the host does not support XON/XOFF, the maximum speed you can use depends on host input buffer size and system load. If a transfer fails at a given baud rate, reduce the baud rate until the transfer is successful.

Examples

1. The file RELSYS.SAV is transferred from an RT–11 host system to the file RELSYS.SAV on an RT–11 stand-alone system:

   ```
   .TRANSF RELSYS.SAV=RELSYS.SAV/W RET
   ```

2. This command transfers the file SYSBLD.COM from a stand-alone system to a file named SYSBLD.COM on a host running RT–11:

   ```
   .TRANSF SYSBLD.COM=DW:SYSBLD.COM/T RET
   ```

TRANSF Confirmation Messages

TRANSF, when used with the /W option, confirms the start of the transfer by displaying this message:

```
Creating [TT::]<outfile> from [TT::]<infile>.
```

where:

TT::	specifies the stand-alone system, and appears with the input or output file specification, depending on how you have issued the command.
outfile	specifies the file (the device, file name, and file type) being created.
infile	specifies the file (the device, file name, and file type) being copied.

If you choose either the /W or /S option, TRANSF prints the following information when the file transfer is complete:

- Number of blocks transferred and number of retries.
- Number of characters saved through compression coding. (Compression coding enables TRANSF to transfer data using fewer characters than normal, which saves transfer time.)
- Confirmation of file transfer.

Example

```
.TRANSF REL12.MAC=REL12.MAC/T/W
Creating REL12.MAC from TT::REL12.MAC
 10 blocks transferred with 0 retries
1198 characters saved through compression encoding
REL12.MAC created from TT::REL12.MAC
```

This example shows a typical file transfer, from a stand-alone system to a host.

Chapter 30

The Virtual Run Utility (VBGEXE)

The Virtual Run Utility (VBGEXE) is an unsupported utility that creates a pseudo unmapped-monitor environment enabling you to run programs faster and with less low-memory space than the programs would otherwise require. VBGEXE lets you execute, without relinking, many well-behaved programs as if they were operating under an unmapped monitor.

The name of the utility is an acronym for *Virtual Background Execution*. VBGEXE is called virtual since it appears to extend the amount of low memory available under a mapped monitor.

Why Use VBGEXE

If you are running under a mapped monitor and there is not enough low memory for your program to execute, try using VBGEXE. VBGEXE uses a small portion of low memory and maps the rest of the job into high memory. As a result, VBGEXE allows you to run programs which might otherwise take up more low-memory space than you have available. Also, VBGEXE tends to speed up the execution of a job.

A program run using VBGEXE is referred to as a completely virtual job or as running in a completely virtual environment.

Programs You Can Run with VBGEXE

You can run the MACRO–11 assembler, the linker, and most RT–11 programs, using VBGEXE. A few RT–11 programs, such as IND, SPOOL, RESORC, and RTMON do not work with VBGEXE. If a job is not a valid one for VBGEXE, that job does not run and VBGEXE issues the error message:

```
?VBGEXE-F-Cannot run in completely virtual environment <dev:filnam.typ>
```

Automatically Running Jobs Under VBGEXE

The DCL commands V and VRUN allow VBGEXE to run programs under mapped monitors. See the *RT–11 Commands Manual* for descriptions of these commands.

If you want your programs to run in a completely virtual environment without having to use the V or VRUN commands, include the command SET RUN VBGEXE in your STRTxx.COM file (or just issue it), if you are running under a mapped monitor.

If you issue the SET RUN VBGEXE command, then:

- If you use R or RUN (or V or VRUN) and the job or environment is valid for running under VBGEXE, the job runs as a completely virtual job.
- If you use R or RUN and the job or environment is not valid for running under VBGEXE, the monitor attempts to run the job as a straight background job.
- If you use V or VRUN and the job or environment is not valid for running under VBGEXE, VBGEXE issues an error message and the job does not run.
- If you issue an FRUN or SRUN command, the specified job is run as a foreground or a system job. The SET RUN VBGEXE command does not apply to foreground or system jobs.

See the description of the SET RUN [NO]VBGEXE command in the *RT–11 Commands Manual* for further information.

Displaying the RUN State (VBGEXE or NOVBGEXE)

Issue the SHOW CONFIGURATION command to get a system display showing whether RUN is set to VBGEXE or NOVBGEXE.

The RUN command runs a job using VBGEXE when all the following conditions are true:

- The file you want to run is on the specified disk.
- VBGEXE.SAV is on SY.
- The command SET RUN VBGEXE is in effect.
- The VBGEX$ (200) bit is set in the job's $JSX word (offset 4 in block 0 of the SAV file header). See the *RT–11 Volume and File Formats Manual* for a description of the $JSX word.

If VBGEXE.SAV is not present on SY when an automatic run is attempted, the message *?KMON-F-File not found SY:VBGEXE.SAV* is displayed.

Running Background, Foreground, or System Jobs

You can use VBGEXE to run background, foreground, or system jobs.

Running a Background Job Using VBGEXE

The following three examples show three different ways of running the MACRO–11 assembler in the background as a completely virtual job. The progam to be compiled is MYPROG.MAC. The object file is to be named MYPROG.OBJ and the listing file, MYPROG.LST.

1. In the first example, V invokes the assembler which then prompts for CSI command-line input:

   ```
   .V MACRO  RET
   *MYPROG,MYPROG=MYPROG  RET
   *
   ```

2. In the second example, V uses DCL command-line syntax to issue the same command as the preceding one. In this case, since V is a DCL command, RT–11 returns you to the keyboard monitor dot prompt when it finishes execution:

   ```
   .V MACRO MYPROG MYPROG  RET
   .
   ```

3. In the third example, the SET RUN command enables VBGEXE to automatically load and execute programs in a completely vitual environment. Then the MACRO command automatically compiles MYPROG in a virtual environment without it being necessary to use the V command:

   ```
   .SET RUN VBGEXE
   .MACRO MYPROG/LIST:MYPROG
   ```

Running VBGEXE as a Foreground or System Job

You can also run VBGEXE as a foreground or system job, and even assign it to its own terminal. In addition, if your monitor includes system-job support, you can use the /NAME option to specify the program you want VBGEXE to run. For example, the following command tells VBGEXE to execute BASIC at terminal 1:

```
.SRUN SY:VBGEXE.SAV/NAME:BASIC/TERM:1  RET
```

How VBGEXE Allocates and Uses Low Memory

VBGEXE allocates and uses low memory for background or foreground and system jobs in the following manner:

- For a background job, VBGEXE allocates all free low memory to the monitor to satisfy user requests (.CDFN, .QSET, .FETCH) that require low memory.
- For a foreground or system job, VBGEXE establishes an approximate 1185 (decimal) word buffer in low memory. About 759 (decimal) words of that buffer are available to the monitor to satisfy user requests (.CDFN and .QSET) that require low memory. The buffer is large enough to let VBGEXE run most jobs.

Using the /BUFF option

If your job cannot run under VBGEXE because the .CDFN or .QSET programmed request requires more low memory, use the /BUFF:nnnnn option in the following manner:

```
.FRUN SY:VBGEXE.SAV/BUFF:nnnnn[/NAME:xxxxxx[/option]] RET
```

or

```
.SRUN SY:VBGEXE.SAV/BUFF:nnnnn[/NAME:xxxxxx[/option]] RET
```

where nnnnn is a decimal value for the number of additional words you want VBGEXE to allocate as buffer space for that job. Note that increasing the buffer size decreases the available low memory.

Running Separated I-and-D Space Programs

RT–11 uses VBGEXE to load and run separated I-and-D (Instruction and Data) space programs.

Loading Separated I-and-D Space Programs

VBGEXE loads separated I-and-D space programs by loading the image into extended memory in a single local region created by VBGEXE at load time specifically for this purpose. VBGEXE loads only the root segment of D and I spaces at load time. It uses the D-space bitmap in block 0 to determine which data blocks to load. It uses the I-space bitmap in the I-space of virtual block 0 to determine which instruction blocks to load. It also places a RETURN instruction (207_8) in the first word of each I space /V overlay partition. The job's D space is set up to contain virtual vectors and the SYSCOM area. The job's I-space is set up to contain the following code in addresses 0 and 2 to facilitate proper completion routine operation:

```
BIC     R0,R0
.ASTX
```

VBGEXE relies on the overlay handler to load other parts of the program and to mark /O overlays as being non-resident.

Running a Separated I-and-D Space Program

Once VBGEXE has loaded a separated I-and-D space program, the job is started running based on its transfer address stored in location 40 of the I-space virtual block 0 of the SAV file. The following programmed requests behave differently when issued from a separated I-and-D space job than from a traditional RT–11 job. However, they behave similarly to how traditional jobs run under VBGEXE behave.

1. .FETCH works from background jobs only. If the handler requested is not already resident (or loaded), it is fetched into kernel memory below the USR.
2. .CDFN allocates channels in kernel memory below the USR and free of PAR1 space.
3. .QSET allocates queue elements in kernel memory below the USR and free of PAR1 space.

Restrictions for Using VBGEXE

The following restrictions apply to the programs VBGEXE can run:

- The program cannot contain interrupt service routines.
- The program cannot use the following programmed requests:

  .FORK
  .INTEN
  .MFPS
  .MTPS
- The program should use only the .PEEK, .POKE, .GVAL, and .PVAL programmed requests to access the kernel address space (for example, monitor data and the I/O page).
- The program can always gain direct access to the kernel by attaching and mapping the global regions KERNEL and IOPAGE.
- The program can initially have direct access to the I/O page by setting the IOPAG$ (40) bit in the job's $JSX word (offset 4 in block 0 of the SAV file header).

Chapter 31

Virtual Terminal Communications Utility (VTCOM)

The Virtual Terminal Communications Utility (VTCOM) enables you to connect your stand-alone RT–11 operating system to another computer's operating system and to communicate between the two operating systems. VTCOM is called the Virtual Terminal Communications utility since it also enables you to use your RT–11 terminal as if it were a terminal of the operating system to which you are connected. In this situation, your RT–11 computer is called the *local* computer, and the computer to which you are connected is called the *host* computer — since it acts as a host, letting you use its resources.

With VTCOM, you have available the resources of the host system such as electronic mail and programming languages, in addition to still being able to use the resources of the stand-alone RT–11 system. VTCOM will transfer ASCII files between the host and the stand-alone RT–11 system.

Users of the VTCOM utility should read the communications chapter in the features section of the *Introduction to RT–11*. This chapter describes how to configure all the components you need (hardware and software) to successfully communicate between a computer running RT–11 and another computer.

VTCOM Functionality

VTCOM does the following:

- Allows you to log onto a host computer and use all the software and hardware facilities (for example, electronic mail and programming languages) on that computer, in addition to all the resources on your local computer.
- Allows you to open on your local computer a logging file that records everything transmitted from the host computer to your local computer's terminal screen.
- Supports some commands and customizations that let you send, for example, a particular dial string to your modem.
- Adjusts the transfer rate of information based on the success or failure it has in transferring that information. (Both VTCOM and TRANSFER/TRANSF do this.)
    - Reduces the transmission packet size by half for each error that occurs during a data transmission. Transmission errors successively halve the packet size from a maximum of 512 bytes to a minimum of 16 bytes.
    - Progressively restores packet size during continued success from transmitting at a reduced packet size. This process is dynamic; errors halve packet

size, while success progressively restores packet size. The maximum and minimum packet sizes are 512 and 16 bytes respectively.

The process of packet-size reduction and restoration is controlled by the program sending the data. Data transfers from the RT–11 standalone processor to a host processor are controlled by VTCOM. Data transfers from a host processor to the RT–11 standalone processor are controlled by TRANSF.

VTCOM sends data at the interface interrupt speed, but if that transfer speed is too fast for the host terminal service to process, you can slow down the baud rate or you can let VTCOM adjust the transfer rate, using retries and reduced packet size. A symptom of a too fast transfer rate is a beeping terminal.

- Manages all file operations on your local computer when you use the native transfer utility (TRANSFER/TRANSF) installed on the host (see Chapter 29 to transfer files between your local computer and the host).
- Allows you to transfer ASCII files between your local computer and a host, but this process is slow and without error checking. If you have the native transfer utility installed on the host, use that utility to transfer files.

VTCOM Package

The VTCOM package consists of three components:

- The VTCOM utility
- The handler that enables you to use VTCOM
- The native-file transfer utility (TRANSFER/TRANSF) that enables you to copy files between computers

The following list describes these components:

- **VTCOM Utility**

  RT–11 distributes two versions of VTCOM. Which version you use depends on the RT–11 monitor you are running at your local computer. The following table lists the monitors with the corresponding version of VTCOM that runs on that monitor.

RT–11 Monitor	Version of VTCOM
Mapped monitor	VTCOM.SAV (for running VTCOM as a system job)
Unmapped multi-job monitor	VTCOM.REL (for running VTCOM as a foreground job)

- **VTCOM Device Handler**

  This handler manages the communications port on the local computer. Any serial line of type DL, DZ, or DH can serve as the communications port for the VTCOM utility. The following table shows the correct communications device handler file for each computer and monitor combination.

Computer	Monitor	Device Handler
CTI-bus processor	Mapped monitor	XCX.SYS
CTI-bus processor	Unmapped multi-job monitor	XC.SYS
PDP–11	Mapped monitor	XLX.SYS
PDP–11	Unmapped multi-job monitor	XL.SYS

  Note that VTCOM for RT–11 V5.2 and later cannot be used with earlier versions of the XC and XL handlers; the XC and XL handlers for V5.2 and later cannot be used with earlier versions of VTCOM.

  Note also that to unload the XC or XL handler, you must first pause or exit from VTCOM.

- **Native-File Transfer Utility (TRANSFER/TRANSF)**

  The native-file transfer utility installed on the host enables you to transfer files between computers. The following table matches the appropriate native-file transfer utility with the host operating system on which it can run. See Chapter 29 for a description of this utility.

Transfer Utility	**Host Operating System**
TRANSF.EXE	VMS
TRANSF.TSK	RSX
TRANSF.SAV	RT–11

Running VTCOM

You can run VTCOM under any monitor. However, VTCOM requires that your monitor include timer support. If you want to run VTCOM under the SB monitor, you must generate a special monitor to include timer support.

- **To run VTCOM as a foreground or system job:**

  1. Load VTCOM into memory, but do not run it. For example:

     ```
     .FRUN VTCOM.REL/PAUSE RET
     ```

     or

     ```
     .SRUN VTCOM.SAV/PAUSE RET
     ```

     These commands assume that VTCOM.REL or VTCOM.SAV is on your system volume. Otherwise, include the volume's device mnemonic in the VTCOM file specification.

  2. Load the handler and assign it exclusive ownership of VTCOM. For example:

     ```
     .LOAD XL=F RET    (if you plan to run VTCOM as a foreground job)
     ```

     or

     ```
     .LOAD XL=VTCOM RET    (if you plan to run VTCOM as a system job)
     ```

  3. Run VTCOM. For example:

     ```
     .RESUME VTCOM RET
     ```

  These command sequences are included in the distributed start-up command files. To implement a command sequence, edit out the exclamation-point comment delimiters; for example:

  ```
  SRUN VTCOM.SAV/PAUSE
  LOAD XL=VTCOM
  RESUME VTCOM
  ```

  Note that you must explicitly exit from VTCOM before you can unload the XL or XC handler.

- **To run VTCOM as a background job:**

  1. Load the handler. For example:

     ```
     .LOAD XL RET
     ```

  2. Run VTCOM. For example:

     ```
     .RUN ddn:VTCOM RET
     ```

     where ddn represents the device on which VTCOM.SAV is located.

Using VTCOM

Once VTCOM is running, you use it to connect your local computer to a host computer and to communicate between the two computers. To do so, you should know how to:

- Call VTCOM
- Establish a link with a host
- Log onto a host computer through VTCOM
- Move between the host and the local computer
- Move between the host/local computer and VTCOM
- Use VTCOM commands
- Use TRANSFER/TRANSF to copy files between computers

The rest of this chapter is divided into six sections covering all but the last of the preceding topics. See Chapter 29 for a description of the TRANSFER/TRANSF utilities.

Calling VTCOM

Once VTCOM is running, how you call it (connect your terminal to that job) depends on how you run VTCOM:

- To call VTCOM as a system job:
  1. Press CTRL/X.
  2. At the system job prompt (Job?), type VTCOM and press RET. For example:

     ```
     .CTRL/X
     Job? VTCOM  RET
     ```

- To call VTCOM as a foreground job:
  1. Press CTRL/F.
  2. At the foreground job prompt (F>), press RET. For example:

     ```
     .CTRL/F
     F>  RET
     ```

Establishing a Link with a Host

Once you are connected with the VTCOM utility, you can establish a link with your host system.

- If your stand-alone system is connected to the host by a hard-wired connection, the link will be established just by calling VTCOM.
- If you plan to communicate with the host over a telephone line, you must dial a number to establish a link. Follow the instructions provided for your modem. If VTCOM is configured for your modem, you can use the VTCOM DIAL command to dial the number. In that case, once you have called VTCOM and have received the VTCOM> prompt, do the following:

  1. Press CRTL/P to enter VTCOM command mode.
  2. At the TT::VTCOM> prompt, issue the DIAL command. See Table 31–1 for further information.

  Note that if you log on to a host computer through a telephone line, be sure to log off the system before breaking the telephone link.

Logging On to a Host

Once linked by hard-wire or telephone lines, you can log on to a host computer through VTCOM.

- If you have a hard-wired link:

  1. At the VTCOM> prompt, press CTRL/P.
  2. At the TT::VTCOM> prompt, type BREAK and press RET.
  3. Continue to press RET until you receive a connection or log-on prompt from the host computer.
  4. Log onto the host computer as you would from a directly attached console terminal.

- If you have a telephone link, follow the instructions for your modem. When you receive a connection or log-on prompt from the host computer, log onto the host computer as you would from a directly attached console terminal.

Moving Between the Host and the Local Computer

When you are logged onto a host computer through VTCOM, VTCOM is in *Terminal Mode*. When in terminal mode, you can move back and forth between the host computer and the local computer, if you are running VTCOM as a system or foreground job.

- To return to your local computer, press CTRL/B. The prompt B> appears. Then press RET and the keyboard monitor prompt (.) appears. For example:

  ```
  CTRL/B
  B> RET
  .
  ```

- To return to the host computer:
  - If you are running VTCOM as a system job, press CTRL/X. The `Job?` prompt appears. Then type `VTCOM` RET. For example:

    ```
    . CTRL/X
    Job? VTCOM  RET
    ```

  - If you are running VTCOM as a foreground job, press CTRL/F. The `F>` prompt appears. Then press RET. For example:

    ```
    .CTRL/F
    F> RET
    ```

Entering and Exiting VTCOM Command Mode

VTCOM allows you to do other things besides logging into a host computer. However, to issue VTCOM commands, you must enter VTCOM command mode. You can do so by pressing CTRL/P either when logged on a host or when you have called VTCOM. This gives you the VTCOM prompt TT::VTCOM. For example:

```
CTRL/P
TT::VTCOM>
```

You can change the character that must be typed to enter VTCOM command mode. To do so, see the customization patch in the *RT–11 Installation Guide* on changing the VTCOM command-mode character.

To return to the host from VTCOM command mode, type CONTINUE and press RET; for example:

```
CONTINUE  RET
```

To log on the host from VTCOM command mode, Type BREAK and press RET; for example:

```
TT::VTCOM> BREAK  RET
```

Continue to press RET until you receive a connection or log-on prompt from the host computer. Then log onto the host computer.

VTCOM Command Summary

To issue a VTCOM command, first enter terminal mode by typing CTRL/F or CTRL/X and the system job name. Then, at the VTCOM> prompt, type CTRL/P to enter VTCOM command mode. VTCOM prompts:

```
TT::VTCOM>
```

Type any of the commands listed in Table 31–1 and press RET. The shortest valid abbreviation for each command is underlined. You can display a list of VTCOM commands on your terminal by typing the VTCOM command HELP or by pressing RET in response to the TT::VTCOM prompt.

Table 31–1: VTCOM Command Descriptions

Command	Function
^x	Lets VTCOM transmit CTRL characters that would normally be intercepted: CTRL/B, CTRL/F, CTRL/O, CTRL/P, CTRL/Q, CTRL/S.
BREAK	Transmits a break signal to the host, as if you had pressed the BREAK key.
CLEAR	Clears any CTRL/S characters that have been sent, and starts sending characters to the terminal again.
CLOSELOG	Stops recording input in a log file and closes the log file. Use this command to make a log file permanent when you have finished transferring a file from the host to your stand-alone system.
CONTINUE	Returns your system to terminal mode. Use this command to exit VTCOM command mode and continue communication with the host system.
CTRL/P	Sends a CTRL/P character to the host, VTCOM normally intercepts CTRL/P characters and interprets them as a request to enter a VTCOM command.
DIAL	Causes the modem to dial the telephone dial string you specify. When you type the DIAL command and press RET, VTCOM prompts you for a string of numbers, letters, or symbols: `TT::VTCOM>Dial string?` Type the string you want the modem to dial and press RET. VTCOM remembers this number for future DIAL commands until you dial a new number, exit VTCOM, or reboot the system.

Table 31–1 (Cont.): VTCOM Command Descriptions

Command	Function
	The VTCOM autodialing feature uses the prefix and suffix characters that are appropriate for the DF224 modem. The default prefix is <CTRL/A>. The default suffix is an exclamation mark (!). See the customization patches in the *RT–11 Installation Guide* for how to modify the default prefix and suffix characters. Apply the appropriate software customization provided in the *RT–11 Installation Guide* to set a default telephone dial string.
EXIT	Terminates the VTCOM program and the XC or XL handler and closes any open logging file. To restart VTCOM, you must use the FRUN or SRUN command.
FAST	Lets VTCOM transmit ASCII characters to the host at high speed during a SEND operation. This command is valid only if the host system supports XON/XOFF for its input service.
HANGUP	Breaks the modem connection. VTCOM drops the DTR signal, holds it low for 2 seconds to break the connection, and then raises it. HANGUP has the same effect as setting the DATA/TALK button on a modem to TALK for 2 seconds, then restoring it to DATA.
HELP	Prints a list of VTCOM commands on your console.
LOG	Resumes recording data in a log file after a NOLOG command.
NOLOG	Suspends the recording of data in a log file. If you are transferring a file from a host to your stand-alone system, the transfer continues and information will be lost.
OPENLOG	Opens a log file to receive ASCII input from the host system, and starts recording input in the log file. You can have only one log file open at a time. If you try to open a second log file, VTCOM closes the first log file before opening the new one.
PAUSE	Ends VTCOM program control, but leaves the XL or XC handler running to receive input from the host.
RESET	Halts file transfers using TRANSF and VTCOM SEND operations. RESET does not halt the VTCOM OPENLOG operation, and does not halt logging.
SELECT	Lets you specify a port on a Mini-Exchange. For information on using VTCOM with the Mini-Exchange, see the Using the Mini-Exchange section.

Table 31–1 (Cont.): VTCOM Command Descriptions

Command	Function
	Type SELECT RET. VTCOM displays a prompt. In response to the prompt, specify a Mini-Exchange port. Valid port identifiers are the numbers 1–8 and the letters M or R. M and R both indicate that port 8 is connected to a modem.
SEND	Transfers an ASCII file from your stand-alone system to a host as if the file were being typed. The VTCOM SEND command sends ASCII files at two speeds: SLOW or FAST. The distributed default speed is SLOW. Use SLOW if the host terminal service does not support XON/XOFF and FAST if it does support XON/XOFF. A customization in the *RT–11 Installation Guide* lets you set the VTCOM SEND command speed.

SHOW	Displays status of the following VTCOM characteristics:
	Data transfers in progress
	Logging status—on or off
	SEND status—slow or fast
	Current dial string
	For example:
	`Packets sent      = 4`
	`Packets received  = 3`
	`Packet size       = 256`
	`Next active block = 3`
	`Logging is OFF`
	`SEND is SLOW`
	`Dial string is not set`

SLOW	Causes VTCOM to transmit ASCII characters to the host at slow speed during a SEND operation. This is useful when the host's terminal service does not support XON/XOFF. This is the default. Apply the appropriate software customization provided in the *Introduction to RT–11* to set FAST as the default speed.

Capturing the Host’s Screen Image

You can capture the screen image from the host computer and transfer that image to a file on the local computer. To do so, do the following:

1. While logged into the host, type CTRL/P to enter VTCOM command mode.
2. At the VTCOM prompt, type the following command:

   ```
   TT::VTCOM> OPENLOG RET
   ```

3. VTCOM prompts you for the name and type of file on your stand-alone system to which you want to send the screen image. Type the file specification.

   ```
   TT::VTCOM> Log File?  myfile.log RET
   ```

   When you have done the preceding you automatically exit VTCOM command mode and are back on the host system.
4. Display on the host’s screen whatever information you want to capture in your stand-alone’s file.
5. When you have displayed all the information you need, type CTRL/P to again enter VTCOM command mode.
6. At the VTCOM prompt, enter the following command to close your log file:

   ```
   TT::VTCOM> CLOSELOG RET
   ```

   This completes the screen capture and leaves you back on the host system.

Copying ASCII Files to and from the Host

The native-file transfer utility (TRANSFER/TRANSF) is especially designed to transfer files between a host and a local computer. However, you can also use VTCOM by itself to transfer files.

The following two sections show how you can use the VTCOM OPENLOG and SEND commands to transfer ASCII files between a host and a local computer. Use this method of transferring files only if you do not have TRANSFER/TRANSF installed on the host.

Copying ASCII Files from the Host (OPENLOG)

The *Introduction to RT–11* has a good example of how to create log files of work displayed on a host terminal. This section shows you how to use that same functionality to copy an ASCII file from the host.

Begin by running VTCOM and establishing a link to your host system (see the section *Running VTCOM* and the section *Establishing a Link with a Host*). Then, when you have logged onto the host system, follow these steps:

1. Type:

   ```
   TYPE filnam.typ
   ```

   filnam.typ represents the name and type of the file you want copied to your stand-alone system. Do not press RET.

2. Type CTRL/P to enter command mode, and type the OPENLOG command:

   ```
   TT::VTCOM> OPENLOG  RET
   ```

3. VTCOM prompts you for the name and type of the file on your stand-alone system to which you want to send the host file. Type the file specification.

   ```
   TT::VTCOM> Log File? filnam.typ  RET
   ```

   This completes the OPENLOG command, and VTCOM leaves command mode.

4. Press RET once again. VTCOM begins to transfer the file. As the file transfers, it is displayed on the screen.

5. When VTCOM finishes sending the file (the file finishes scrolling on the screen and the host system prompt appears), enter VTCOM command mode once again by typing CTRL/P.

6. Type the CLOSELOG command and press RET. This closes the newly created file on your stand-alone system.

   ```
   TT::VTCOM> CLOSELOG  RET
   ```

The file on your stand-alone system will contain extra characters transmitted from the host: a carriage return, line-feed combination at the beginning of the file, and the host system’s prompt character at the end of the file. Delete these extra characters by editing the file with a text editor such as KED.

Suppressing Form-Feed and Tab Conversions

Because some terminals cannot process the form-feed (FF) character, the host processor terminal service in some systems converts an embedded form-feed character into a carriage return/multiple line-feed combination before sending a file. Tabs are often converted in the same way.

To suppress these character conversions, issue the following command on the VMS or RSX host system before starting a data transfer.

```
SET TERM/FORM/TAB RET
```

You can include this command in a log-in or start-up command file on your host system. You do not need to issue the command on your stand-alone system.

Copying ASCII Files to the Host (SEND)

Begin by running VTCOM and establishing a link to your host system (see the section *Running VTCOM* and the section *Establishing a Link with a Host*). Then, when you have logged on to the host system, follow these steps:

1. Type the command appropriate for your host's operating system to send terminal input to a file. For example, if your host system is RT–11 or VMS, type:

   ```
   COPY TT: filnam.typ RET
   ```

   filnam.typ represents the name and type of the output file to which you are copying.

2. Type CTRL/P to enter command mode, and type the SEND command:

   ```
   TT::VTCOM> SEND RET
   ```

3. VTCOM prompts you for the name and type of the file you want to send to the host system. Type the file specification for the file you want to send to the host, and press RET.

   ```
   TT::VTCOM> Send File? filnam.typ RET
   ```

   This completes the SEND command, and VTCOM leaves command mode. VTCOM begins to transfer the file. As the file is transferred, it is displayed on your screen.

4. When VTCOM finishes sending the file (the file finishes scrolling on the screen), type CTRL/Z. This closes the newly created file on the host.

Using the Mini-Exchange

The Mini-Exchange is a microprocessor-based communication device that lets personal computers and workstations in an office environment exchange data in a small switched point-to-point network. The Mini-Exchange has eight communication ports and thus can provide up to four concurrent point-to-point connections.

Under VTCOM, you can connect your RT–11 system to a host system through the Mini-Exchange. To establish a connection:

1. (If you are using a Professional 300 series computer, do this step; other computers have the speed set in the hardware.) Set the bit transfer rate using the XC SPEED=n command, where n is one of the following values specifying bits per second:

   300
   600
   1200
   2400
   4800
   9600
   19200

   Other transfer rates are valid once the connection has been established.

2. Issue the VTCOM SELECT command and press RET. VTCOM displays a prompt.

3. In response to the prompt, specify the Mini-Exchange port to which the host system is connected and press RET. Valid identifiers are the numbers 1 to 8 and the letters M and R. (M or R indicates that port 8 is a modem port.)

4. VTCOM responds by displaying the status of the port. The four valid status responses are:

   A = accepted
   B = rejected
   C = no device
   Z = busy

   If the port is currently busy (status Z), the Mini-Exchange queues your request.

If you do not receive a status within 3 seconds after making a port connection request, check for the following conditions:

- The port you selected is not connected to any device.
- The port you selected is your own port to the Mini-Exchange.
- The device you want to connect to is malfunctioning or powered off. Check the device.
- The cable on the device you want to connect to is malfunctioning. Check the cable.

- The port you selected is malfunctioning. Perform the diagnostic tests described in the Mini-Exchange documentation.

The following example requests port 7 of a Mini-Exchange. VTCOM accepts the request and establishes a connection with the processor attached to port 7.

```
TT::VTCOM> SELECT RET
TT::VTCOM> PORT? 7 RET
A
```

The following example requests port 8 as a modem port. VTCOM accepts the request and establishes the modem connection with the processor using port 8.

```
TT::VTCOM> SELECT RET
TT::VTCOM> PORT? M RET
A
```

Appendix A BATCH

RT–11 BATCH is a complete job-control language that allows RT–11 to operate unattended. BATCH processing is ideally suited to frequently run production jobs, large and long-running programs, and programs that require little or no interaction with you, the user. With BATCH, you can prepare your job on any RT–11 input device and leave it for the operator to start and run.

RT–11 BATCH permits you to:

- Execute an RT–11 BATCH stream from any RT–11 input device.
- Output a log file to any RT–11 output device (except magtape or cassette).
- Execute the BATCH stream with a single-job monitor or in the background with a multi-job monitor.
- Generate and support system-independent BATCH language jobs.
- Execute RT–11 monitor commands from the BATCH stream.

RT–11 BATCH consists of the BATCH compiler and the BATCH run-time handler. The BATCH compiler reads the batch input stream you create, translates it into a format suitable for the RT–11 BATCH run-time handler, and stores it in a file. The BATCH run-time handler executes this file with the RT–11 monitor. As each command in the batch stream executes, BATCH lists the command, along with any terminal output generated, by executing the command on the BATCH log device.

Hardware and Software Requirements

You can run RT–11 BATCH on any single-job foreground/background or extended-memory system that is configured with at least 16K words of memory. A line printer, although optional, is highly desirable as the log device.

BATCH uses certain RT–11 system programs to perform its operations. For example, the $BASIC command executes the file BASIC.SAV. Make sure that the following RT–11 programs are on the system device, with exactly the following names, before you run BATCH:

BASIC.SAV	(BASIC users only)
BA.SYS	
BATCH.SAV	
CREF.SAV	(MACRO users only)
SYSLIB.OBJ	(FORTRAN and MACRO users)
FORTRA.SAV	(FORTRAN users only)
LINK.SAV	
MACRO.SAV	(MACRO users only)
PIP.SAV	
DIR.SAV	

Control-Statement Format

For input to RT–11 BATCH, you can generate a file with the RT–11 editor and use any RT–11 input device. The input consists of BATCH control statements. A BATCH control statement is divided into three fields, separated from one another by spaces: command fields, specification fields, and comment fields. The control statement has the syntax:

$command/option   specification/option   [!comment]

Each control statement requires a specific combination of command and specification fields and options (see the Commands section). Control statements cannot be longer than 80 characters, excluding multiple spaces, tabs, and comments. You can use a hyphen (-) as a line continuation character to indicate that the control statement is continued on the next line (see Table A-4). Even if you use the line continuation character, the maximum control statement length is still 80 characters.

The following example of a $FORTRAN command illustrates the various fields in a control statement:

```
$FORTRAN/LIST/RUN  PROGA/LIBRARY  PROGB/EXE  !RUN FORTRAN
 command/options      spec fields/options   comment field
```

Command Fields

The command field in a BATCH control statement indicates the operation to be performed. It consists of a command name and certain command field options. Indicate the command field with a $ in the first character position and terminate it with a space, tab, blank, or return.

Command Names

The command name must appear first in a BATCH control statement and have a dollar sign ($) in the first position of the command (for example, $JOB). No intervening spaces are allowed in the command name. BATCH recognizes only two forms of a command name: the full name, and an abbreviation consisting of $ and the first three characters of the command name. For example, you can enter the $FORTRAN command as:

```
$FORTRAN
```

or

```
$FOR
```

You cannot enter it as:

```
$FORT
```

or

```
$FORTR
```

Command-Field Options

Options that appear in a command field are command qualifiers. Their functions apply to the entire control statement. All option names must begin with a slash (/) that immediately follows the command name. Table A–1 describes the command field options for BATCH and indicates the commands on which you can use them. Those option characters that appear in square brackets are optional. The command field options are described in greater detail in the sections dealing with the appropriate commands.

NOTE

All /NO options are the defaults, except the /WAIT option in the $MOUNT and $DISMOUNT commands and the /OBJECT option in the $LINK command.

Table A–1: Command Field Options

Option	Function
/BAN[NER]	Prints the header of the job on the log file. BATCH allows this option only on the $JOB command. Note that BATCH outputs the $JOB command line to the log device sixty times.
/NOBAN[NER]	Does not print a job header.
/CRE[F]	Produces a cross-reference listing during compilation. BATCH allows this option only on the $MACRO command.
/NOCRE[F]	Does not create a cross-reference listing.
/DEL[ETE]	Deletes input files after the operation completes. BATCH allows this option on the $COPY and $PRINT commands.
/NODEL[ETE]	Does not delete input files after operation completes.
/DOL[LARS]	The data following this command can have a $ in the first character position of a line. BATCH allows this option on the $CREATE, $DATA, $FORTRAN, and $MACRO commands. BATCH terminates reading data when you use one of the following commands or when it encounters a physical end-of-file on the BATCH input stream: `$JOB` `$EOD` `$SEQUENCE` `$EOJ`
/NODOL[LARS]	The data following this command cannot have a $ in the first character position; a $ in the first character position means a BATCH control command.
/LIB[RARY]	Includes the default library in the link operation. BATCH allows this option on the $LINK and $MACRO commands.
/NOLIB[RARY]	Does not include the default library in the link operation.
/LIS[T]	Produces a temporary listing file (see the Temporary Files section) on the listing device (LST) or writes data images on the log device (LOG). BATCH allows this option on the $BASIC, $CREATE, $DATA, $FORTRAN, $JOB, and $MACRO commands. When you use /LIST on the $JOB command, /LIST sends data lines in the job stream to the log device (LOG).
/NOLIS[T]	Does not produce a temporary listing file.
/MAP	Produces a temporary link map on the listing device (LST). BATCH allows this option on the $FORTRAN, $LINK, and $MACRO commands.
/NOMAP	Does not create a MAP file.
/OBJ[ECT]	Produces a temporary object file as output from compilation or assembly (see the Temporary Files section). BATCH allows this option on the $FORTRAN, $LINK, and $MACRO commands. When you use /OBJECT on $LINK, BATCH includes temporary files in the link operation.

Table A–1 (Cont.): Command Field Options

Option	Function
/NOOBJ[ECT]	Does not produce an object file as output of compilation; with $LINK, does not include temporary files in the link operation.
/RT11	Sets BATCH to operate in RT–11 mode (see the RT–11 Mode section). BATCH allows this option only on the $JOB command.
/NORT11	Does not set BATCH to operate in RT–11 mode.
/RUN	Links (if necessary) and executes programs compiled since the last link-and-go operation or start of job. BATCH allows this option on the $BASIC, $FORTRAN, $LINK, and $MACRO commands.
/NORUN	Does not execute or link and execute the program after performing the specified command.
/TIM[E]	Writes the time of day to the log file when BATCH executes. BATCH allows this option only on the $JOB command. This command writes the time after each command that begins with a dollar sign ($).
/NOTIM[E]	Does not write the time of day to the log file.
/UNI[QUE]	Checks for unique spelling of options and keynames (see the $JOB command section). BATCH allows this option only on the $JOB command.
/NOUNI[QUE]	Does not check for unique spelling.
/WAI[T]	Pauses for operator action. BATCH allows this option on the $DISMOUNT, $MESSAGE, and $MOUNT commands.
/NOWAI[T]	Does not pause for operator action.
/WRI[TE]	Indicates that the operator is to WRITE-ENABLE a specified device or volume. BATCH allows this option only on the $MOUNT command.
/NOWRI[TE]	Indicates that no writes are allowed or that the specified volume is read-only; informs the operator, who must WRITE-LOCK the appropriate device.

Specification Fields

Specification fields immediately follow command fields in a BATCH control statement and apply only to the fields they follow. Use them to name the devices and files involved in the command. You must separate these fields from the command field, and from each other, by blanks or spaces.

If a specification field contains more than one file to be used in the same operation, separate the files by a plus (+) sign. For example, to assemble files F1 and F2 to produce an object file F3 and a temporary listing file, type:

```
$MACRO/LIST F1+F2/SOURCE F3/OBJECT
```

If you need to repeat a command for more than one field specification, separate the files by a comma (,). For example, the following command assembles F1 to produce F2, a temporary listing file, and a map file F3. It then assembles F4 and F5 to produce F6 and a temporary listing file:

```
$MACRO/LIST F1/SOURCE F2/OBJECT F3/MAP,F4+F5/SOURCE-
F6/OBJECT
```

Depending on the command you use, specification fields can contain a device specification, file specification, or an arbitrary ASCII string. You can use an appropriate specification field option (see Table A–3) with any of these three items.

Physical Device Names

Represent each device in an RT–11 BATCH specification field with a standard two- or three-character device name. If you do not specify a unit number for devices that have more than one unit, BATCH assumes unit 0.

In addition to physical device names, you can assign logical device names to devices. A logical device name takes precedence over a physical name, thus providing device independence. With this feature, you do not need to rewrite a program that is coded to use a specific device if the device is unavailable. For example, DK is initially assigned to the system device, but you can assign that name to diskette unit 1 (DX1) with an RT–11 monitor ASSIGN command.

You must assign certain logical names prior to running any BATCH job. BATCH uses these logical names as default devices. These names are:

LOG	BATCH log device (cannot be magtape or cassette)
LST	Default for listing files generated by BATCH stream

The following are not legal device names in RT–11; if you use them, the operator must assign them as logical names with the ASSIGN command. You can use these names in BATCH streams written for other Digital systems.

DF	Fixed-head disk (RF)
LL	Line printer with uppercase and lowercase characters
M7	7-track magtape

M9	9-track magtape
PS	Public storage (DK as assigned by RT–11)

See the ASSIGN keyboard command in the *RT–11 Commands Manual* for instructions on assigning logical names to devices.

File Specifications

You can reference files symbolically in a BATCH control statement with a name of up to six alphanumeric characters followed, optionally, by a period and a file type of three alphanumeric characters. Tabs and embedded spaces are not allowed in either the file name or file type. The file type generally indicates the format of a file. It is good practice to conform to the standard file types for RT–11 BATCH. If you do not specify a file type for an output file, BATCH and most other RT–11 system programs assign appropriate default file types. If you do not specify a file type for an input file, the system searches for that file name with a default file type. Table A–2 lists the standard file types used in RT–11 BATCH.

Table A–2: BATCH File Types

File Type	Explanation
BAS	BASIC source file (BASIC input)
BAT	BATCH command file
CTL	BATCH control file generated by the BATCH compiler
CTT	BATCH temporary file generated by the BATCH compiler
DAT	BASIC or FORTRAN data file
DIR	Directory listing file
FOR	FORTRAN IV source file (FORTRAN input)
LST	Listing file
LOG	BATCH log file
MAC	MACRO source file (MACRO or SRCCOM input)
MAP	Link map output from $LINK operation
OBJ	Object file output from compilation or assembly
SOU	Temporary source file
SAV	Runnable file or program image output from $LINK

Wildcard Construction

You may use wildcards in certain BATCH control statements (such as $COPY, $CREATE, $DELETE, $DIRECTORY, $PRINT). You can use the asterisk as a wildcard to designate the entire file name or file type.

NOTE

You cannot use embedded wild cards (* or %) in BATCH control statements. However, you can use them in the keyboard monitor commands if you use the RT–11 mode of BATCH.

Specification Field Options

Specification field options follow file specifications in a BATCH control statement and designate how the file will be used. These options apply only to the field in which they appear. Option names begin with a slash. The specification field options for RT–11 BATCH are listed in Table A–3. Optional characters in the option names are in square brackets.

Table A–3: Specification Field Options

Option	Explanation
/BAS[IC]	BASIC source file.
/EXE[CUTABLE]	Indicates the executable program image file to be created as the result of a link operation.
/FOR[TRAN]	FORTRAN source file.
/INP[UT]	Input file; default if you specify no options.
/LIB[RARY]	Library file to be included in link operation (prior to default library).
/LIS[T]	Listing file.
/LOG[ICAL]	Indicates that the device is a logical device name; use in $DISMOUNT and $MOUNT commands.
/MAC[RO]	MACRO source file.
/MAP	Linker map file.
/OBJ[ECT]	Object file (output of assembly or compilation).
/OUT[PUT]	Output file.
/PHY[SICAL]	Indicates physical device name.
/SOU[RCE]	Indicates source file.
/VID	Volume identification.

Comment Fields

Comment fields, which document a BATCH stream, are identified by an exclamation point (!) appearing anywhere except in the first character position of the control statement. BATCH treats any character following the ! and preceding the return/line feed combination as a comment. For example:

```
$RUN PIP     !DELETE FILES ON DK:
```

This command runs the RT–11 system program PIP. BATCH ignores the comment.

You can also include comments as separate comment lines by typing a $ in character position 1, followed immediately by the ! operator and the comment. For example:

```
$!DELETE FILES ON DK:
```

BATCH Character Set

The RT–11 BATCH character set is limited to the 64 uppercase characters (ASCII 40 through 137). The current ASCII set is assumed (character 137 is underscore and not left-arrow, and character 136 is circumflex, not up-arrow). The BATCH job-control language does not support any control characters other than tab, return, and line feed.

Table A–4 shows how BATCH normally interprets certain characters. Character interpretations are different if you use RT–11 mode (see The RT–11 Mode section)).

Table A–4: Character Explanation

Character	Explanation
space	Specification field delimiter. It separates arguments in control statements. BATCH considers any string of consecutive spaces and tabs (except in quoted strings) as a blank (that is, equivalent to a single space).
!	Comment delimiter. The input routine ignores all characters after the exclamation point, up to the return/line feed combination.
"	Passes a text string containing delimiting characters where the normal precedence rules would create the wrong action. For example, use it to include a space in a volume identification (/VID).
$	BATCH control statement recognition character. A dollar sign ($) in the first character position of a BATCH input stream line indicates that the line is a control statement.
.	Delimiter for file type.

Table A–4 (Cont.): Character Explanation

Character	Explanation
-	Indicates line continuation if the character after the hyphen is one of the following: • A return/line feed • Any number of spaces or tabs followed by a return/line feed • A comment delimiter (!) • Spaces followed by a comment delimiter (!) If any other character follows the hyphen, the hyphen is assumed to be a minus sign indicating a negative value in an option.
/	Precedes an option name. An alphanumeric string must immediately follow it.
0–9	Numeric string components.
:	Immediately follows a device name. You can also use it to separate an option name from its value or to separate an option value from its subvalue (you can use : interchangeably with = for this purpose).
A–Z	Alphabetic string components.
=	Separates an option name from a value.
\	Illegal character except when it precedes a directive to the BATCH run-time handler from the operator (see the Communicating with BATCH jobs section). (To include \ in an RT–11 mode command, use \\.)
+	File delimiter. Separates multiple files in a single specification field. Also indicates a positive value in options.
,	Separates sets of arguments for which the command is to be repeated.
*	A wildcard in utility command file specifications.
CR/LF	return/line feed. It indicates end-of-line (or end of logical record) for records in the BATCH input stream.

Temporary Files

When you do not include field specifications in a BATCH command line, BATCH sometimes generates temporary files. For example, you can enter a $FORTRAN command that is followed in the BATCH stream by the FORTRAN source program as:

```
$FORTRAN/RUN/OBJECT/LIST
 FORTRAN source program
$EOD
```

This command generates a temporary source file from the source statements that follow, a temporary object file, a temporary listing file, and a temporary memory image file.

BATCH sends temporary files to the default device (DK) or the listing device (LST) according to their type. If the device is file-structured, BATCH assigns file names and file types as follows:

nnnmmm.LST	for temporary listing files (sent to LST)
nnnmmm.MAP	for temporary map files (sent to LST)
nnnppp.OBJ	for temporary object files (sent to DK)
000000.SAV	for temporary memory image files (sent to DK)
nnnppp.SOU	for temporary source files (sent to DK)

where:

nnn	represents the last three digits of the sequence number assigned to the job by the $SEQUENCE command (see the $SEQUENCE command section). Thus, a sequence number of 12345 produces a file name beginning 345. If you do not use the $SEQUENCE command, BATCH sets nnn to 000.
mmm	represents the number of listing (or map) files BATCH generated since the BATCH run-time handler (BA.SYS) was loaded. The first such file, listing or map, is 000. Each time BATCH generates a new temporary file, it increments the file name by 1. Thus, the second listing file produced under job sequence number 12345 is 345001.LST, and the first map file produced is 345000.MAP.
ppp	represents the number of object or source files in the current BATCH run. The first such file (object or source) is 000. Each time BATCH generates a new temporary file, it increments the file name by 1. BATCH resets these file names to 000 every time you run BATCH and after every $LINK, $MACRO, or $FORTRAN command that uses the temporary files.

General Rules and Conventions

You must adhere to the following general rules and conventions associated with RT–11 BATCH processing:

- Always place a dollar sign ($) in the first character position of a command line.
- Each job must have a $JOB and $EOJ command.
- You can spell out command and option names entirely or you can specify only the first three characters of the command and required characters of the option.
- Specify wildcard construction (*) only for the utility commands ($COPY, $CREATE, $DELETE, $DIRECTORY, and $PRINT) and for commands that normally accept wildcards in RT–11 mode.
- Include comments at the end of command lines or in a separate comment line. When you include comments in a command line, place them after the command but precede them by an exclamation mark.
- Include only 80 characters per control statement, excluding multiple spaces, tabs, and comments.
- When you omit file specifications from BATCH commands and supply data in the BATCH stream, the system creates a temporary file with a default name (see the Temporary Files section).
- You can use the RT–11 monitor type-ahead feature only with BATCH handler directives (see Communicating with BATCH Jobs section) to be inserted into a BATCH program. No other terminal input (except input to a foreground program) can be entered while a BATCH stream is executing.
- You cannot use an indirect command file to call BATCH.

Commands

Place BATCH commands in the input stream to indicate to the system which functions to perform in the job. All BATCH commands have a dollar sign ($) in the first character position (for example, $JOB). Intervening spaces are not allowed in command names. The command name must always start in the first character position of the line.

BATCH commands are presented in alphabetical order in this chapter for ease of reference. However, if you are not familiar with BATCH, read the commands in a functional order as listed in Table A–5. The characters shown in square brackets are optional.

Table A–5: BATCH Commands

Command	Function
$SEQ[UENCE]	Assigns an arbitrary identification number to a job.
$JOB	Indicates the start of a job.
$EOJ	Indicates the end of a job.
$MOU[NT]	Signals the operator to mount a volume on a device and optionally assigns a logical device name.
$DIS[MOUNT]	Signals the operator to dismount a volume from a device and deassigns a logical device name.
$FOR[TRAN]	Compiles a FORTRAN source program.
$BAS[IC]	Compiles a BASIC source program.
$MAC[RO]	Assembles a MACRO source program.
$LIB[RARY]	Specifies libraries for BATCH to use in link operations.
$LIN[K]	Links modules for execution.
$RUN	Causes a program to execute.
$CAL[L]	Transfers control to another BATCH file, executes that BATCH file, and returns to the calling BATCH stream.
$CHA[IN]	Passes control to another BATCH file.
$DAT[A]	Indicates the start of data.
$EOD	Indicates the end of data.
$MES[SAGE]	Issues a message to the operator.
$COP[Y]	Copies files.
$CRE[ATE]	Creates new files from data included in the BATCH stream.
$DEL[ETE]	Deletes files.
$DIR[ECTORY]	Provides a directory of the specified device.

Table A–5 (Cont.): BATCH Commands

Command	Function
$PRI[NT]	Prints files.
$RT[11]	Specifies that the following lines are RT–11 mode commands.

For each command listed below, the term filespec represents a device name, or file name, and a file type. Filespec has this form:

```
dev:filnam.typ
```

As a general rule, BATCH assumes device DK if you omit a device specification.

$BASIC

The $BASIC command calls RT–11 single-user BASIC to execute a BASIC source program. The $BASIC command has the following syntax:

$BASIC[/option...] [filespec/option] [!comments]

where:

/option Indicates an option you can append to the $BASIC command. The options are as follows:

/RUN	indicates that BATCH should execute the source program.
/NORUN	indicates that BATCH should only compile the program and send error messages to the log file.
/LIST	writes data images that are contained in the job stream to the log file (LOG).
/NOLIST	writes data images to the log file only if you specify $JOB /LIST.

filespec indicates the name and type of the source file and the device on which it resides. If you omit the file type, BATCH assumes BAS. If you omit this specification, the source statements must immediately follow the $BASIC command in the input stream.

Terminate the source program after a $BASIC statement with either a $EOD command or with any other BATCH command that starts with a $ in the first position.

/option indicates an option that can follow the source file name. BATCH assumes any file name with no option appended is the name of a source file. This option can have one of the following values (or you can omit it):

/BASIC	indicates that the file name you specify is a BASIC source program.
/SOURCE	performs the same function as /BASIC.
/INPUT	performs the same function as /BASIC.

You can follow the $BASIC command with the source program, BASIC commands (such as RUN), or data. The following two BATCH streams, for example, produce the same results (but BATCH does not echo the same output format for both streams).

```
$BASIC              $BASIC/RUN
10 INPUT A          10 INPUT A
20 PRINT A          20 PRINT A
30 END              30 END
RUN                 $DATA
123                 123
$EOD                $EOD
```

$CALL

The $CALL command transfers control to another BATCH control file, temporarily suspending execution of the current control file. BATCH executes the called file until it reaches $EOJ or until the job aborts; control then returns to the statement following the $CALL in the originating BATCH control file. You can nest calls up to 31 levels. BATCH includes the log file for the called file in the log file for the originating BATCH program. (See NOTE following the $EOJ command.)

The syntax of the $CALL command is:

$CALL filespec[!comments]

Options are not allowed in the $CALL command. BATCH saves $JOB command options across a $CALL; however, they do not apply to the called BATCH file. If you specify CTL as the file type, BATCH assumes a precompiled BATCH control file. If you do not specify a file type, BATCH assumes .BAT and compiles the called BATCH stream before execution.

NOTE

If the called program generates temporary files, those files can supersede existing temporary files if the two jobs have the same sequence number. For example, consider the following two BATCH streams:

```
$FOR/OBJ A      $FOR/OBJ A
$FOR/OBJ B      $CALL C
$LINK/RUN       $FOR/OBJ B
```

The called BATCH file (C.BAT) contains the following:

```
$JOB
$FOR/OBJ A1
$FOR/OBJ B1
$LINK/RUN
$EOJ
```

The temporary object files C.BAT generates change the behavior of the previous two BATCH statement sequences. The first temporary file created by C.BAT (000000.OBJ) supersedes the temporary file produced by the first $FORTRAN command (000000.OBJ). You can avoid this situation by giving the BATCH job C.BAT a unique sequence number (see the $SEQUENCE command section).

$CHAIN

The $CHAIN command transfers control to a named BATCH control file but does not return to the input stream that executed the $CHAIN command. The syntax of the $CHAIN command is:

$CHAIN filespec[!comments]

BATCH does not permit options in the $CHAIN command. If you specify CTL as the file type, BATCH assumes a precompiled BATCH control file. If you do not specify a file type, BATCH assumes .BAT and compiles the chained BATCH stream before execution.

A $EOJ command should always follow the $CHAIN command in the BATCH stream.

NOTE

The values of BATCH run-time variables remain constant across a $CALL, $CHAIN, or return from call. See the Creating RT–11 Mode BATCH Programs section for a description of these variables.

Use the $CHAIN command to transfer control to programs that you need to run only once at the end of a BATCH stream. For example, you could use the following BATCH program (PRINT.BAT) to print and then delete all temporary listing files generated during the current BATCH job:

```
$JOB                    !PRINT ALL LIST FILES
$PRINT/DELETE *.LST
$EOJ
```

You could then run PRINT.BAT with the $CHAIN command as follows:

```
$JOB
$MACRO/RUN       A ALST/LIST
$MACRO/RUN       B BLST/LIST
$CHAIN PRINT
$EOJ
```

$COPY

The $COPY command copies files in image mode from one device to another. You can use the wildcard construction (see the Specification fields section) in the input and output file specifications. You can concatenate several input files to form one output file (as long as the output specification does not contain a wildcard). The $COPY command has the following syntax:

$COPY[/option] output-filespec[...,output-filespec]/OUTPUT-
input-filespec[...,input-filespec][/INPUT][!comments]

where:

/option	indicates options that you can append to the $COPY command.	
	/DELETE	deletes input files after the copy operation.
	/NODELETE	does not delete input files after the copy operation.
output-filespec	represents an output file; you must specify a file type.	
	/OUTPUT	indicates that a file specification is for an output file.
input-filespec	represents a file to be copied. (BATCH copies files to the output file in the order that you list them, except when you use wildcards.)	
	/INPUT	indicates that a file specification is for an input file; if you do not specify an option, BATCH assumes INPUT.

The following are examples of the $COPY command:

```
$COPY *.BAS/OUTPUT DL1:*.BAS
```

This command copies all files with the file type BAS from the volume on unit 1 to the default storage device DK:

```
$COPY FILE2.FOR/OUTPUT FILE0.FOR+FILE1.FOR
```

This command merges the input files FILE0.FOR and FILE1.FOR to form one file called FILE2.FOR and stores FILE2.FOR on device DK:

```
$COPY *.*/OUT DL0:*.FOR, DL1:*.*/OUT DL0:*.*
```

This command copies all files with the file type FOR from DL0 to DK and all files on DL0 to DL1.

$CREATE

The $CREATE command generates a file from data records that follow the $CREATE command in the input stream. An error occurs if the data does not immediately follow the $CREATE command. You cannot precede the data records with a $DATA command.

You can follow the $CREATE data with a $EOD command to signify the end of data, or you can use any other BATCH control statement to indicate end of data and initiate a new function. The $CREATE command has the following syntax:

$CREATE[/option...] filespec [!comments]

where:

/option	indicates an option you can append to the $CREATE command. The options are:

/DOLLARS	indicates that the data following this command can have a $ in the first character position of a line.
/NODOLLARS	indicates that a $ cannot be in the first character position of a line.
/LIST	writes data image lines to the log file.
/NOLIST	does not write data image lines to the log file. If you specify $JOB/LIST, BATCH ignores this option.

filespec	represents the file you want to create.

NOTE

If you use the /DOLLARS option, you must follow the last data record with a $EOD command (see Table A–1).

The following is an example of the $CREATE command:

```
$CREATE/LIST PROG.FOR
FORTRAN source file
$EOD
```

The data records following the $CREATE command become a new file (PROG.FOR) on the default device (DK). BATCH generates a listing on logical device LOG.

$DATA

Use the $DATA command to include data records in the input stream. Data you include in this manner needs no file name. BATCH transfers the data to the appropriate program as though it were input from the console terminal. For example, you can follow the $RUN command for a particular program by a $DATA command and the data records for the program to process. The data records must be valid data for the program that is to use them.

The $DATA command has the following syntax:

$DATA[/option...] [!comments]

Four options that you can use with the $DATA command are as follows:

/DOLLARS	Indicates that the data following this command can have a $ in the first character position of a line.
/NODOLLARS	Indicates that a $ cannot be in the first character position of a line.
/LIST	Writes data image lines to the log file.
/NOLIST	Does not write data images to the log file. If you specify $JOB /LIST, BATCH ignores this option.

NOTE

Any command beginning with a $ normally follows the last data record. However, if you specify $DATA /DOLLARS, you must follow the last data record with $EOD.

The following example shows data entered into a BASIC program (TEST1.BAS):

```
$BASIC/RUN TEST1.BAS
$DATA
25,75,125,146
180,210,520,874
$EOD
```

Using $DATA with FORTRAN Programs

When you use the $DATA command to provide input to a FORTRAN program, you must insert a CTRL/Z into the BATCH file after the last data line and before $EOD (or before the next BATCH command if you do not use $EOD). This procedure permits FORTRAN to properly detect an end-of-file after it reads the last data line. For example:

```
$FORTRAN/RUN  A.FOR
$DATA
1
2
3
^Z RET  LF
$EOD
$RUN PIP
```

The above program reads three numbers from the input stream and then detects an end-of-file when it attempts to read a fourth number. If you include an END=n statement in your FORTRAN program, statement n gets control when the end-of-file is detected. If the CTRL/Z <RET> <LF> is not present, the program aborts when it reaches $EOD and never executes the END=n statement.

$DELETE

Use the $DELETE command to delete files from the device you specify. This command has the syntax:

$DELETE filespec[...,filespec][!comments]

where:

filespec	represents the name of a file to be deleted.

The following example deletes all files named TEST1 on the default device DK:

```
$DELETE TEST1.*
```

The following example deletes all files with FOR file types on DL1:, then deletes all files with MAC file types on DK:

```
$DELETE DL1:*.FOR,*.MAC
```

$DIRECTORY

The $DIRECTORY command outputs a directory of the device you specify to a listing file. If you do not specify a listing file, the listing goes to the BATCH log file. This command has the syntax:

$DIRECTORY [filespec/LIST] [filespec[...,filespec]][/INPUT]
[!comments]

where:

filespec/LIST	indicates the name of the directory listing file
filespec/INPUT	indicates the input files to be included in the directory (default)

The following command outputs a directory of the device DK to the BATCH log file:

```
$DIRECTORY
```

This next command creates on the device DK a directory file (FOR.DIR) that contains the names, lengths, and dates of creation of all FORTRAN source files on that device.

```
$DIRECTORY FOR.DIR/LIST *.FOR
```

$DISMOUNT

The $DISMOUNT command removes the logical device name assigned by a $MOUNT command. When BATCH encounters $DISMOUNT while executing a job, it prints the entire $DISMOUNT command line on the console terminal. This message tells the operator which device to unload. This command has the syntax:

$DISMOUNT[/option] logical-device-name:[/LOGICAL] [!comments]

where:

/option	indicates an option you can append to the $DISMOUNT command. The options are:

/WAIT	indicates that the job must pause until the operator enters a response. If you do not specify either /WAIT or /NOWAIT, BATCH assumes /WAIT. BATCH rings a bell at the terminal, prints the physical device name to be dismounted followed by a question mark (?), and waits for a response. (At this point you can enter input to the BATCH handler. See the Communicating with BATCH Jobs section)
/NOWAIT	does not pause for operator response; BATCH prints the physical device name to be dismounted.

logical-device-name:	is the logical device name to be deassigned from the physical device.

/LOGICAL	identifies the device specification as a logical device name.

The following example instructs the operator to dismount the physical device with the logical device name OUT and removes the logical assignment of device OUT. In this example, OUT is DL0. The operator dismounts DL0: and then types a return:

```
$DISMOUNT/WAIT OUT:/LOGICAL
DL0?
```

$EOD

The $EOD command indicates the end-of-data record or the end of a source program in the job stream. The syntax of this command is:

$EOD [!comments]

The $EOD command can signal the end of data associated with any of the following commands:

$BASIC	$FORTRAN
$CREATE	$MACRO
$DATA	

In the following example, the $EOD command indicates the end of a source program that is to be compiled, linked, and executed:

```
$FORTRAN/RUN
source program
$EOD
```

$EOJ

The $EOJ command indicates the end of a job. This command must be the last statement in every BATCH job. The command has the following syntax:

$EOJ [!comments]

If BATCH encounters a $JOB command, a $SEQUENCE command, or a physical end-of-file in the input stream before $EOJ, an error message appears in the log file.

NOTE

Make sure that the $EOJ command is the last line in a BATCH file.

$FORTRAN

The $FORTRAN command calls the FORTRAN compiler to compile a source program. Optionally, this command can provide printed listings or list files and can produce a link map in the listing. The $FORTRAN command has the following syntax:

$FORTRAN[/option...] [source-filespec[/option]]
[filespec/OBJECT]-
[filespec/LIST] [filespec/EXECUTE]-
[filespec/MAP] [filespec/LIBRARY] [!comments]

where:

/option indicates an option you can append to the $FORTRAN command. The options are as follows:

/RUN	indicates that FORTRAN is to compile the source program, link it with the default library, and execute it. The default library is SYSLIB.OBJ. You can change it with the $LIBRARY command.
/NORUN	compiles the program only.
/OBJECT	produces a temporary object file.
/NOOBJECT	does not produce a temporary object file.
/LIST	produces a list file on the listing device (LST).
/NOLIST	does not produce a list file.
/MAP	produces a link map on the listing device (LST).
/NOMAP	does not create a MAP file.
/DOLLARS	indicates that the data following this command can have a $ in the first character position of a line.
/NODOLLARS	indicates that a $ cannot be in the first character position of a line.

source-filespec	indicates the device, file name, and file type of the FORTRAN source file. If you do not specify the file name, the $FORTRAN source statements must immediately follow the $FORTRAN command in the input stream; BATCH generates a temporary source file that it deletes after FORTRAN compiles the temporary source file (see the Temporary Files section). You can terminate the source program included after a $FORTRAN statement by either a $EOD command or by any other BATCH command. If, however, you use dollar signs in the first position in the source program, you must enter the source program with $CREATE/DOLLARS. In this case, you cannot use $FORTRAN/DOLLARS.
/option	represents an option that can have one of the following values: **/FORTRAN** indicates that the file name you specify is a FORTRAN source program. BATCH assumes that any file name with no option appended is the name of a source file. **/SOURCE** performs the same function as /FORTRAN. **/INPUT** performs the same function as /FORTRAN.
filespec/OBJECT	indicates the device, file name, and file type of the object file produced by compilation. The object file remains on the device you specify after the job finishes. You must follow the object file specification, if you include it, with the /OBJECT option. If you omit the object file specification but specify $FORTRAN /OBJECT, BATCH creates a temporary object file. BATCH includes this temporary file in any $LINK operations that follow it in the job, and deletes it after the link operation.
filespec/LIST	indicates the name you assign to the list file created by the compiler. BATCH does not automatically print the list file if you assign LST to a file-structured device, but you can list it using the $PRINT command. Follow the list file specification with the /LIST option.
filespec/EXECUTE	indicates the name you assign to a memory image file. Follow the memory image file specification with the /EXECUTE option. If you do not include this field, BATCH generates a temporary memory image file (see the Temporary Files section) and then deletes the temporary file.
filespec/MAP	indicates the name you assign to the link map file created by the linker. Follow the map specification with the /MAP option.
filespec/LIBRARY	indicates that BATCH must include the file you specify in the link procedure as a library before SYSLIB.OBJ. The file must be a library file (produced by the RT–11 librarian). Follow the library specification with the /LIBRARY option.

The following command calls FORTRAN to compile and execute a source program named PROGA.FOR:

```
$FORTRAN/RUN PROGA.FOR
```

The next command sequence compiles the FORTRAN program but does not produce an object file. BATCH creates a temporary listing file on LST:

```
$FORTRAN/NOOBJ/LIST
```

source program

$EOD

NOTE

See the $DATA command section for instructions on using the $DATA command with FORTRAN programs.

$JOB

The $JOB command indicates the beginning of a job. Each job must have its own $JOB command. This command has the following syntax:

$JOB[/option...] [!comments]

BATCH allows the following options in the $JOB command:

/BANNER	Prints a header (a repetition of the $JOB line) on the log file.
/NOBANNER	Does not print a job header.
/LIST	Writes data image lines that are contained in the job stream to the log file.
/NOLIST	Writes data image lines to the log file only when a /LIST option exists on a $BASIC, $CREATE, or $DATA command that has data lines following it.
/RT11	If no $ appears in column 1 when BATCH expects one, BATCH assumes that the line is an RT–11 mode command (see the $CREATE command section).
/NORT11	Does not process RT–11 mode commands.
/TIME	Writes the time of day to the log file when BATCH executes command lines (except $DATA command lines).
/NOTIME	Does not write the time of day.
/UNIQUE	Checks for unique spelling of options and keynames. When you use this option, you can abbreviate commands and options to the fewest number of characters that still make their names unique. For example, you can abbreviate the /DOLLARS option to /DO since no other option begins with the characters DO.
/NOUNIQUE	Checks only for normal option and keyname spellings.

End each job with a $EOJ command if you want to run it. If an input stream consists of more than one job, BATCH automatically terminates one job when it encounters the $JOB command for the next job. BATCH will never run a job terminated with another $JOB command; instead, an error message will appear in the log.

The following $JOB command writes the time of day to the log file before BATCH executes each command beginning with a $. It also accepts unique abbreviations of BATCH commands and options:

```
$JOB/TIME/UNIQUE
```

$LIBRARY

The $LIBRARY command lets you specify a list of library files for inclusion in FORTRAN links or other link operations that have the /LIBRARY option. By default, the list of libraries contains only SYSLIB.OBJ, the RT–11 system library. This command has the syntax:

$LIBRARY filespec [!comments]

or

$LIBRARY filespec+SYSLIB [!comments]

where:

filespec	represents a library file; the default file type is OBJ.
SYSLIB	is the RT–11 system library that you create at system generation.

Libraries are linked in order of their appearance in the $LIBRARY command.

The following example shows two libraries (LIB1.OBJ and LIB2.OBJ) that are included in FORTRAN links before SYSLIB.OBJ:

```
$LIBRARY LIB1.OBJ+LIB2.OBJ+SYSLIB.OBJ
```

$LINK

Use the $LINK command to produce memory image files from object files. This command links any files you may specify with any temporary object files created since the last link or link-and-go operation.

Temporary object files are those files you create as a result of a $FORTRAN or $MACRO command without naming an object file (with the /OBJECT option) by suppressing an object file (with the /NOOBJECT option). Create permanent object files by using the /OBJECT option on a $FORTRAN or $MACRO file descriptor.

BATCH links files in the following order:

1. Temporary files—in the order in which they were compiled
2. Permanent files—in the order in which they are specified in the $LINK command
3. Any library specified by the $LINK command—provided that unresolved references remain
4. The default library list—if you specified $LINK/LIBRARY

The syntax for this command is:

$LINK[/option...] [filespec/OBJECT] [filespec/LIBRARY]-
[filespec/MAP] [filespec/EXECUTE] [!comments]

where:

/option	indicates an option that you can append to the $LINK command. The options are as follows:

/LIBRARY	includes the RT–11 system library (SYSLIB.OBJ) and any default libraries specified in the $LIBRARY command in this $LINK operation. Use this option when the files being linked do not include any temporary FORTRAN object files. You can also use it when you specify $FORTRAN without the /RUN or /MAP option, but want to search the default library list for unresolved references.
/NOLIBRARY	does not include the default libraries.
/MAP	produces a temporary load map on the listing device (LST).
/NOMAP	does not produce a map file.
/OBJECT	includes temporary object files in the link. If you specify neither /OBJECT nor /NO-OBJECT, BATCH assumes $LINK /OBJECT.
/NOOBJECT	does not include temporary files in the link.
/RUN	executes the memory image files associated with this $LINK command when the link is complete.
/NORUN	only links the program and does not execute it.

filespec/OBJECT	indicates the name of the object file BATCH must link; if you do not specify /OBJECT, BATCH assumes it as the default.
filespec/LIBRARY	indicates that the file you specify is to be included in the link procedure as a library; the file you specify must be a library file (produced by the RT–11 librarian).
filespec/MAP	indicates the load map file BATCH must create as a result of the $LINK command.
filespec/EXECUTE	indicates the memory image file BATCH must create as a result of the $LINK command.

The following command links all temporary object files created since the last $LINK command, or the last $FORTRAN/OBJ or $MACRO/OBJ command:

```
$LINK/RUN
```

The next command links the temporary files and the object files PROG1.OBJ and PROG2.OBJ to form a memory image file named PROGA.SAV. It also creates and outputs a temporary map file:

```
$LINK/MAP PROG1.OBJ+PROG2.OBJ/OBJ PROGA.SAV/EXE
```

$MACRO

The $MACRO command calls the MACRO assembler to assemble a source program and, optionally, to provide printed listings or list files. You must specify any MACRO listing directives in the source program; you cannot enter them at BATCH command level.

The $MACRO command has the following syntax:

$MACRO[/option...] [source-filespec[/option]] [filespec/OBJECT]-
[filespec/LIST] [filespec/MAP] [filespec/LIBRARY]-
[filespec/EXECUTE] [!comments]

where:

/option indicates an option you can append to the $MACRO command. The options are as follows:

/RUN	assembles, links, and runs the source program.
/NORUN	only assembles the source program.
/OBJECT	produces a temporary object file.
/NOOBJECT	does not produce a temporary object file.
/LIST	produces a listing file on the listing device (LST).
/NOLIST	does not produce a list file.
/CREF	produces a cross-reference listing during assembly.
/NOCREF	does not produce a cross-reference listing during assembly.
/MAP	produces a link map as part of the listing file on LST.
/NOMAP	does not create a MAP file.
/DOLLARS	indicates that the data following this command can have a $ in the first character position of a line.
/NODOLLARS	indicates that a $ cannot be in the first character position of a line.
/LIBRARY	includes the default library in the link operation.
/NOLIBRARY	does not include the default library in the link operation.

source-filespec	indicates the name of the source file. If you do not specify a file name, the $MACRO source statements must immediately follow the $MACRO command in the input stream. You can terminate the source program you include after a $MACRO statement with either a $EOD command or any other BATCH command. If, however, you include dollar signs in the first position in the source program, use the $CREATE /DOLLARS command to enter the source program. In this case, you cannot use $MACRO/DOLLARS.
/option	can have one of the following values:

/MACRO	indicates that the file name you specify is a MACRO source program. BATCH assumes that any file name with no option appended is the name of a source file.
/SOURCE	performs the same function as /MACRO.
/INPUT	performs the same function as /MACRO.

filespec/OBJECT	indicates the name you assign to the object file produced by compilation. The object file remains on the device you specify after the job finishes. If you include an object file specification, follow it with the /OBJECT option. If you omit the object file specification but specify $MACRO /OBJECT, BATCH creates a temporary object file. BATCH also includes the temporary object file in any $LINK operations that follow the $MACRO command in the job, and deletes it after the link operation (see the Temporary Files section).
filespec/LIST	indicates the name you assign to the list file created by the assembler. BATCH does not print the list file if you assign LST to a file-structured device, but you can list it using the $PRINT command. The /LIST option must follow the list file specification.
filespec/MAP	indicates the file to which BATCH must output the storage map.
filespec/LIBRARY	indicates that BATCH must include the file you specify in the link procedure as a library. The /LIBRARY option must follow the library file specification.
filespec/EXECUTE	indicates the name you assign to a memory image file. The /EXECUTE option must follow the memory image file specification. If you do not include this field but do use $MACRO/RUN, BATCH generates and runs a temporary memory image file (see the Temporary Files section).

The following $MACRO command assembles a program named PROG0.MAC, and creates a temporary object file and a temporary listing file:

```
$MACRO/LIST/OBJECT PROG0.MAC
```

$MESSAGE

Use the $MESSAGE command to issue a message to the operator at the console terminal. It provides a means for the job to communicate with the operator. The $MESSAGE command has the syntax:

$MESSAGE[/option] message [!comments]

where:

/option	indicates an option you can append to the $MESSAGE command. The options are:	
	/WAIT	indicates that the job is to pause until the operator either types a return to continue or enters commands to the BATCH handler followed by a return (see the Communicating With BATCH Jobs section).
	/NOWAIT	does not pause for operator response.
message	is a string of characters that must fit on one console line. BATCH prints the message on the console.	

For example, if you include the following message in the input stream:

```
$MESSAGE/WAIT MOUNT SCRATCH TAPE ON MT0:
```

The message:

```
MOUNT SCRATCH TAPE ON MT0:
?
```

appears on the console terminal and a bell sounds. The operator mounts the tape and types return to allow further processing of the job. (See the Communicating with BATCH Jobs section for operator interaction with BATCH.)

NOTE

BATCH compresses multiple spaces and tabs in BATCH command lines; therefore, attempts to format $MESSAGE output with tabs or spaces may not provide you with the desired results.

$MOUNT

The $MOUNT command assigns a logical device name and other characteristics to a physical device. When BATCH encounters $MOUNT during the execution of a job, it prints the entire $MOUNT command line on the console terminal to notify the operator which volume to use.

The $MOUNT command has the syntax:

$MOUNT[/option...] physical-device-name:[/PHYSICAL][/VID=x]
[logical-device-name:/LOGICAL] [!comments]

where:

/option	indicates an option you can append to the $MOUNT command. The options are:

/WAIT	indicates that the job is to pause until the operator enters a response. If you do not specify either /WAIT or /NOWAIT, BATCH assumes /WAIT. BATCH rings a bell, prints the physical device name and a question mark (?), and waits for a response. (The response can consist of input for the BATCH handler; see the Communicating with BATCH Jobs section)
/NOWAIT	does not pause for operator response. BATCH prints the name of the physical device to be mounted.
/WRITE	tells the operator to write-enable the volume.
/NOWRITE	tells the operator to write-protect the volume.

physical-device-name	is required and specifies the physical device name and an optional unit number followed by a colon (for example, DL1:). If you specify a device name without a unit number, the operator can enter one in response to the question mark printed by the $MOUNT command. If you want the operator to supply a unit number, do not use the /NOWAIT option because it assumes unit 0.
/PHYSICAL	identifies the device specification as a physical unit specification. If you do not specify either /PHYSICAL or /LOGICAL, BATCH assumes /PHYSICAL.
/VID=x **/VID="x"**	provides volume identification. The volume identification is the name physically attached to the volume. Include it to help the operator locate the volume. Use this option only on the physical device file specification. If x contains spaces, specify it as “x”.

NOTE

This volume identification is only a visual check for the operator. Make the identification match the visual label on the volume, not the identification that you wrote onto the volume at initialization time with the INIT /VOLUMEID command.

logical-device-name/LOGICAL	is required to identify any logical device name you may assign to the device. The /LOGICAL option must follow the logical device name specification.

The following command instructs the operator to select a unit and mount volume BAT01 on that unit, write-enabled. It informs the operator by printing:

```
$MOUNT/WAIT/WRITE DL:/VID=BAT01 2:/LOGICAL
DL0?
```

The operator selects a unit, mounts volume BAT01 write-enabled, and responds to the question mark by typing the unit number (such as 1) followed by a return. BATCH assigns logical device name 2 to the physical device (in this case DL1:) and proceeds.

If no unit number response is necessary, as this command shows,

```
$MOUNT/WAIT/WRITE DL1: 2:/LOGICAL
```

the operator responds with a return after mounting the volume and write-enabling the device.

$PRINT

Use the $PRINT command to print the contents of the files you specify on the listing device (LST). This command has the syntax:

$PRINT[/option] filespec [...,filespec][/INPUT] [!comments]

where:

/option	indicates an option you can append to the $PRINT command. The options are:
	/DELETE — deletes input files after printing.
	/NODELETE — does not delete input files after printing.
filespec	represents a file to be printed.
/INPUT	indicates that the file is an input file; BATCH assumes /INPUT if you omit it.

The following command prints a listing of files with file type MAC that are stored on default device DK:

```
$PRINT *.MAC
```

The following example creates listing files for the programs A and B, prints the listing files, and then deletes them:

```
$MACRO A.MAC A/LIST
$MACRO B.MAC B/LIST
$PRINT/DELETE A.LST,B.LST
```

$RT11

The $RT11 command allows the BATCH job to communicate directly with the RT–11 system. DIGITAL recommends that you use RT–11 mode if you use BATCH. This command puts BATCH in RT–11 mode until BATCH encounters a line beginning with $. In RT–11 mode, BATCH interprets all data images as commands to the RT–11 monitor, to RT–11 system programs, or to the BATCH run-time system. The $RT11 command has the syntax:

$RT11 [!comments]

See the RT–11 Mode section for a complete description of the RT–11 mode.

$RUN

The $RUN command executes a program for which a memory image file (SAV) was previously created. It can also run RT–11 system programs.

The $RUN command has the syntax:

$RUN filespec [!comments]

where:

filespec	represents the file to be executed. If you omit the file type, BATCH assumes SAV.

For example, if DIR is on DK, you can run DIR to print a directory listing:

```
$RUN DIR
$DATA
LP:=DK:/L
$EOD
```

$SEQUENCE

The $SEQUENCE command is an optional command. If you use it, it must immediately precede a $JOB command. The $SEQUENCE command assigns a job an arbitrary identification number. BATCH assigns the last three characters of a sequence number as the first three characters of a temporary listing or object file (see the RT–11 Mode section). If a sequence number is less than three characters long, BATCH fills it with zeroes on the left.

The syntax of this command is:

$SEQUENCE id [!comments]

where:

id	represents an unsigned decimal number that indicates the identification number of a job

The following are examples of the $SEQUENCE command:

```
$SEQUENCE 3          !SEQUENCE NUMBER IS 003
$JOB

$SEQUENCE 100        !SEQUENCE NUMBER IS 100
$JOB
```

Sample BATCH Stream

The following sample BATCH stream creates a MACRO program, assembles and links that program, and runs the memory image file. It then deletes the object, memory image, and source files it created and prints a directory of DK showing the files the BATCH stream created:

```
$JOB
$MESSAGE         THIS IS AN EXAMPLE BATCH STREAM
$MESSAGE         NOW CREATE A MACRO PROGRAM
$CREATE/LIST     EXAMPL.MAC
.TITLE  EXAMPL FOR BATCH
        .MCALL   .PRINT,.EXIT
START:  .PRINT   #MESSAG
        .EXIT
MESSAG: .ASCIZ   /EXAMPLE MACRO PROGRAM FOR BATCH/
        .END     START
$EOD
$MACRO  EXAMPL EXAMPL/OBJECT EXAMPL/LIST    !ASSEMBLE
$LINK   EXAMPL EXAMPL/EXECUTE               !AND LINK
$PRINT/DELETE EXAMPL.LST
$MESSAGE        RUN THE MACRO PROGRAM
$RUN    EXAMPL                              !AND EXECUTE
$DELETE EXAMPL.OBJ+EXAMPL.SAV+EXAMPL.MAC
$MESSAGE       PRINT A DIRECTORY
$DIRECTORY     DK:EXAMPL.*
$MESSAGE       END OF THE EXAMPLE BATCH STREAM
$EOJ
```

To run this batch stream, type the following commands at the console. BATCH prints the messages:

```
.LOAD BA,LP
.ASSIGN LP:LOG
.ASSIGN LP:LST
.R BATCH
*EXAMPL
 THIS IS AN EXAMPLE BATCH STREAM
 NOW CREATE A MACRO PROGRAM
 RUN THE MACRO PROGRAM
 PRINT A DIRECTORY
 END OF THE EXAMPLE BATCH STREAM

END BATCH
.
```

The preceding sample BATCH stream produces the following log file on the line printer:

NOTE

The amount of free memory and the directory format are variable.

```
$JOB

$MESSAGE          THIS IS AN EXAMPLE BATCH STREAM

$MESSAGE          NOW CREATE A MACRO PROG.

$CREATE/LIST      EXAMPL.MAC
```

```
.TITLE  EXAMPLE FOR BATCH
        .MCALL  .PRINT,.EXIT
START:  .PRINT  #MESSAG
        .EXIT
MESSAG: .ASCIZ  /EXAMPLE MACRO PROGRAM FOR BATCH/
        .EVEN
        .END    START

$EOD

$MACRO  EXAMPL EXAMPL/OBJECT EXAMPL/LIST   !ASSEMBLE

ERRORS DETECTED: 0

EXAMPLE FOR BATCH       MACRO V03.00 21-JUN-77 00:05:29 PAGE 1

1                                       .TITLE  EXAMPLE FOR BATCH
2                                               .MCALL  .PRINT,.EXIT
3 000000                                START:  .PRINT  #MESSAG
4 000006                                        .EXIT
5 000010      105      130      101     MESSAG: .ASCIZ  /EXAMPLE MACRO PROGRAM FOR BATCH/
  000013      115      120      114
  000016      105      040      115
  000021      101      103      122
  000024      117      040      120
  000027      122      117      107
  000032      122      101      115
  000035      040      106      117
  000040      122      040      102
  000043      101      124      103
  000046      110      000
6                                               .EVEN
7 000000'                                       .END    START

EXAMPLE FOR BATCH       MACRO V03.00 21-JUN-77 00:05:29 PAGE 1-1
SYMBOL TABLE

MESSAG  000010R         START   000000R

. ABS.  000000     000
        000050     001
ERRORS DETECTED: 0

VIRTUAL MEMORY USED: 508 WORDS  ( 2 PAGES)
DYNAMIC MEMORY AVAILABLE FOR  48 PAGES
EXAMPL,EXAMPL=EXAMPL

$LINK   EXAMPL EXAMPL/EXECUTE            !AND LINK

$PRINT/DELETE EXAMPL.LST

$MESSAGE        RUN THE MACRO PROGRAM

$RUN            EXAMPL                   !AND EXECUTE

EXAMPLE MACRO PROGRAM FOR BATCH

$DELETE         EXAMPL.OBJ+EXAMPL.SAV+EXAMPL.MAC

$MESSAGE        PRINT A DIRECTORY

$DIRECTORY      DK:EXAMPL.*

 21-JUN-77
EXAMPL.BAK    2 14-JUN-77      EXAMPL.BAT             2 21-JUN-77
EXAMPL.CTL    3 21-JUN-77
 3 FILES, 7 BLOCKS
 1903 FREE BLOCKS

$MESSAGE        END OF THE EXAMPLE BATCH STREAM

$EOJ
```

RT–11 MODE

RT–11 mode lets you enter commands to the RT–11 monitor or to system programs, and lets you create BATCH programs. You can enter RT–11 mode with either the $JOB/RT11 command or the $RT11 command. If you enter RT–11 mode with the $JOB/RT11 command, RT–11 mode remains in effect until BATCH encounters the next $JOB command. If you enter RT–11 mode with the $RT11 command, RT–11 mode is in effect until BATCH encounters a $ in the first position of the command line.

When the characters ., $, *, and tab or space appear in the first position of a line, they are control characters and indicate the following:

. command to the RT–11 monitor, for example:

```
.R PIP
```

* data line; any line not intended to go to the RT–11 monitor or to the BATCH run-time handler, such as a command to the RT–11 PIP program:

```
*FILE1.DAT/D
```

NOTE

BATCH does not pass the * as data to the program. Comment lines (!) cannot appear on data lines, as BATCH would consider them as data.

$ BATCH command. It causes an exit from RT–11 mode if you entered RT–11 mode with the $RT11 command. For example:

```
$RT11                 !ENTER RT--11 MODE
.R PIP
*FILE1.DAT/D
$FORTRAN              !LEAVE RT--11 MODE
```

space/tab separator to indicate a line directed to BATCH run-time handler. This separator is indicated by a TAB in the following descriptions.

Communicating with RT–11

The most common use of RT–11 mode is to send commands to the RT–11 monitor and to run system programs. For example, you can insert the following commands in the BATCH stream to run PIP and save backup copies of files:

```
$RT11
.R PIP
*DL1:*.*=*.FOR
```

You must anticipate and include in the BATCH input stream responses that the called program requires, such as the Y response to DUP's Are you sure? query. Place a line in your BATCH file consisting of Y and RETURN or use the DUP /Y option to suppress the query. For example:

```
$RT11
.INITIALIZE RK1:
*Y
```

You can communicate directly with the RT–11 monitor by using the keyboard monitor commands; for example:

```
$RT11
.DELETE/NOQUERY DX1:*.MAC
```

This command deletes all files with a file type of MAC from device DX1.

You cannot mix BATCH standard commands with RT–11 mode data lines (lines beginning with an asterisk). For example, the proper way to do a $MOUNT within a sequence of RT–11 mode data commands is:

```
$JOB/RT11
.R MACRO
*A1=A1
*A2=A2
$MOUNT DL0:/PHYSICAL
.R MACRO
*B1=DL:B1
*B2=DL:B2
```

Creating RT–11 Mode BATCH Programs

Advanced system programmers can use RT–11 mode to create BATCH programs. These BATCH programs consist of standard RT–11 mode commands (monitor commands, data lines for input to system programs, and so on) plus special RT–11 mode commands. The BATCH run-time handler interprets these special commands to allow dynamic calculations and conditional execution of the RT–11 mode standard commands. The following can help you create BATCH programs and dynamically control their execution at run-time:

- Labels
- Variable modification:
  1. Equating a variable to a constant or character (LET statement)
  2. Passing the value of a variable to a program
  3. Incrementing the value of a variable by 1
  4. Conditional transfers on comparison of variable values with numeric or character values (IF and GOTO statements)
- Commands to control terminal I/O
- Other control characters
- Comments

Labels

You define labels in RT–11 mode to provide a symbolic means of referring to a specific location within a BATCH program. If present, a label must begin in the first character position, must be unique within the first six characters, and must terminate with a colon (:) and a return/line feed combination.

Variables

A variable in RT–11 mode is a symbol representing a value that can change during program execution. The 26 variables BATCH permits in a BATCH program have the names A–Z; each variable requires one byte of physical storage. There are four ways to modify variables.

You can assign values to variables in a LET statement.

You can then test these values by an IF statement to control the direction of program execution.

Assign values to variables with a LET statement of the following form:

TAB **LET x="c**

where:

x	represents a variable name in the range A–Z.
"c	indicates the ASCII value of a character.

For example:

```
TAB LET A="0
```

This example indicates that the value of variable A is the 7-bit ASCII value of the character 0 (60).

The LET statement can also specify an octal value in the form:

TAB **LET A=n**

where:

n	represents an 8-bit signed octal value in the range 0–377. Positive numbers range from 0–177; negative numbers range from 200–377 (–200 to –1).

You can use variables to introduce control characters, such as ESCAPE, into a BATCH stream. For example, wherever 'A' appears in the following BATCH stream, BATCH substitutes the contents of variable A (the code for an ESCAPE):

```
$JOB/RT11
   LET A=33
   !A IS AN ESCAPE
.R EDIT
*EBFILE.MAC'A''A'
*R'A''A'
   !EDIT FILE TO CHANGE THE VERSION NUMBER TO 2
*GVERSION='A'DI2'A''A'
*EX'A''A'
```

Increment the value of a variable by 1 by placing a percentage sign (%) before the variable. For example:

```
TAB %A
```

This command indicates that BATCH must increase the unsigned contents of variable A by 1.

Indicate with an IF statement conditional transfers of control according to the value of a variable. The IF statement has the syntax:

TAB **IF(x-"c) label1, label2, label3**

or

TAB **IF(x-n) label1, label2, label3**

where:

x	represents the variable to be tested.

"c	is the ASCII value to be compared with the contents of the variable.
n	is an octal integer in the range 0–377.
label1 **label2** **label3**	represent the names of labels included in the BATCH stream.

When BATCH evaluates the expression (x–"c) or (x–n), the BATCH run-time handler transfers control to:

- label1, if the value of the expression is less than zero.
- label2, if the value of the expression is equal to zero.
- label3, if the value of the expression is greater than zero.

If you omit one of the labels, and the condition is met for the omitted label, control transfers to the line following the IF statement.

NOTE

Since this comparison is a signed byte comparison, 377 is considered to be –1.

The characters + and - allow you to control where BATCH begins searching for label1, label2, and label3. If you precede the label by a minus sign (-), BATCH starts the label search just after the $JOB command. If a plus sign (+) or no sign precedes the label, the label search starts after the IF statement. For example:

```
TAB IF(B-"9) -LOOP, LOOP1,
```

This statement transfers program control to the label LOOP following the $JOB command if the contents of variable B are less than the ASCII value of 9. It transfers control to the label LOOP1 following the IF statement if B is equal to ASCII 9. If the contents of variable B are greater than the ASCII value of 9, program control goes to the next BATCH statement in sequence.

The GOTO statement unconditionally transfers program control to a label you specify as the argument of the statement. You can use one of the following three forms of this statement:

TAB GOTO label	transfers control to the first occurrence of label that appears after this GOTO statement in the BATCH stream.
TAB GOTO +label	same as GOTO label.
TAB GOTO -label	transfers control to the first occurrence of label that appears after the $JOB command.

The following GOTO statement transfers control unconditionally to the next label LOOP if such a label appears in the BATCH stream following the GOTO statement:

```
TAB GOTO LOOP
```

NOTE

If BATCH cannot find a label (for example, you unintentionally omit a minus sign), the BATCH handler searches until it reaches the end of the CTL file and ends the job.

Terminal I/O Control

You can issue commands directly to the BATCH run-time handler to control logging console terminal input and output. If you do not enter any of the following commands, BATCH assumes TTYOUT (this includes indirect command files):

TAB NOTTY	does not write terminal input and output to the log file. Comments to the log are still logged.
TAB TTYIN	writes only terminal input to the log file.
TAB TTYIO	writes terminal input and output to the log file. (You should enter this command if using RT–11 mode so that RT–11 mode commands go to the log file.)
TAB TTYOUT	writes only terminal output to the log file (default).

Other Control Characters

The system permits other control characters in an RT–11 mode command that begins with a period (.) or an asterisk (*). Following are these control characters and their meanings:

'text' command to BATCH run-time handler, where text can be one of the following:

CTY	accepts input from the console terminal; notifies the operator that action is required by ringing a bell and printing a question mark (?).
FF	outputs the current log buffer.
NL	inserts a new line (line feed) in the BATCH stream.
x	inserts the contents of a variable where x is an alphanumeric variable in the range A through Z. It indicates that BATCH should insert the contents of the variable as an ASCII character at this place in the command string.
"message"	directs the message to the console terminal.

The following commands allow the operator to enter the name of a MACRO program to be assembled. The BATCH stream contains:

```
$JOB/RT11
.R MACRO
*'"ENTER MACRO COMMAND STRING"''CTY'
```

The operator receives the following message at the terminal and types a response, followed by a return; BATCH processing continues:

```
ENTER MACRO COMMAND STRING
?FILE,FILE=FILE
```

To run the same BATCH file on several systems with different configurations you need to assign a device dynamically. The following RT–11 mode command lets you request that the listing device name be entered by the operator:

```
.ASSIGN '"PLEASE TYPE LST DEVICE NAME"''CTY'LST
```

The operator receives the message and responds with the device to be used as the listing device (DL2:):

```
PLEASE TYPE LST DEVICE NAME
?DL2:
```

Comments

You can include comments in RT–11 mode as separate comment statements. Include comments by typing a separator followed by a ! and the comment. For example:

```
TAB !OPERATOR ACTION IS REQUESTED IN THIS JOB. BE PREPARED.
```

RT–11 Mode Examples

The following are examples of BATCH programs using the RT–11 mode.

This BATCH program assembles, lists, and maps 10 programs with only 12 BATCH commands:

```
$JOB/RT11    !ASSEMBLE, LIST, MAP PROG0 to PROG9
   TTYIO
   !WRITE TERMINAL I/O TO THE LOG FILE
   LET N="0
   !START AT FILE PROG0
LOOP:
.R MACRO
*PROG’N’,LOG:/C=PROG’N’/N:TTM
.R LINK
*,LOG:=PROG’N’
   %N
   !INCREMENT VARIABLE N
   IF(N-"9)-LOOP,-LOOP,END
   !TEST FOR END
END:
$EOJ
```

The following program lets you set up a master control stream to run several BATCH jobs with one call to BATCH. First set up a BATCH job (INIT.BAT) that performs a $CHAIN to the master control stream:

```
$JOB/RT11
   LET I="0
   !INITIALIZE INDEX
$CHAIN MASTER        !GO TO MASTER
$EOJ
```

The following is the master control stream (MASTER.BAT) to which INIT chains:

```
$JOB/RT11            !MASTER CONTROL STREAM
   %I
   !BUMP INDEX BY 1
   IF(I-"7),,END
.R BATCH
   !THIS IS A $CHAIN
*JOB’I’
   !RUNS JOB1-JOB7
END:
$MESSAGE END OF BATCH RUN
$EOJ
```

Each job MASTER.BAT will run must contain the following:

```
$JOB
   !BATCH COMMANDS
$CHAIN MASTER
$EOJ
```

Activate the master control stream by calling BATCH as follows:

```
.R BATCH
*INIT
```

Operating Procedures

This section describes the operations you must perform to prepare for using BATCH, and for running BATCH.

Loading BATCH

After you bootstrap the RT–11 system and enter the date and time, you must make the BATCH run-time handler resident by typing the RT–11 LOAD command as follows:

```
.LOAD BA:
```

You detach and unload the BATCH run-time handler with the /U option in the BATCH compiler command line (see the Operating Procedures section).

NOTE

If BATCH crashes, you must unload BATCH with the UNLOAD command and then reload BATCH with the LOAD command. This ensures that the BATCH handler is properly initialized when you rerun BATCH.

You must make the BATCH log device and list device resident unless the log or list device is SY, or unless it is a device for which the handler is already resident. Load the log device, using the following syntax:

.LOAD log-device

where:

log-device	represents the device to which BATCH must write the log file.

For example:

```
.LOAD LP:
```

You can, of course, load device handlers with a single LOAD command. For example:

```
.LOAD BA:,LP:
```

You must then assign the logical device name LOG to the log device. Use the RT–11 monitor ASSIGN command in the form:

.ASSIGN log-device LOG

For example, if LP: is the log device, type:

```
.ASSIGN LP LOG
```

Then assign the logical device name LST using the RT–11 ASSIGN command in the form:

.ASSIGN list-device LST

where:

list-device	represents the physical device BATCH must use for listings.

If, for example, you want to produce listings on the line printer, type:

```
.ASSIGN LP LST
```

NOTE

Do not use the DEASSIGN command with no arguments in a BATCH program since it deassigns the log and list devices, possibly causing the BATCH job to terminate.

You must also make resident the BATCH run-time handler input device (compiler output device). If this device is already resident or is SY, you do not need to load it. For example, to load the DL handler as the input device, type:

```
.LOAD DL
```

Running BATCH

When you have loaded all necessary handlers, run the BATCH compiler as follows:

```
.R BATCH
```

BATCH responds by printing an asterisk (*) to indicate its readiness to accept commands. In response to the *, type the output file specifications for the control file followed by an equal sign. Then type the input file specifications for the BATCH file as follows:

**[[output-filespec][,log-filespec][/option...]=]input-filespec[...,
input-filespec][/option...]**

where:

output-filespec	is the BATCH compiler output device and file the BATCH run-time handler must use. The device you specify must be random-access. Your BATCH job should not delete or move this file. Your BATCH job should avoid compressing the system volume with the SQUEEZE command or the DUP /S option. If you omit output-filespec, BATCH generates a file on the default device DK with the same name as the first input file but with a CTL file type. If you do not specify a file type in output-filespec, BATCH assumes CTL.

log-filespec is the log file created by the BATCH run-time handler. If you do not specify a log device, BATCH assumes LOG. The device name you specify for log-filespec must be the same as you assign to LOG.

You can change the size of a log file on a file-structured device from the default size of 64(decimal) blocks. To make this change, enclose the required size in square brackets. For example:

```
*,FILE.LOG[10]=FILE
```

The default file type for the log-filespec is LOG.

input-filespec represents an input file. If you do not specify a file type, BATCH assumes BAT. If you specify a CTL file, BATCH assumes a precompiled file that must be the only file in the input list.

/option is an option from the following list:

/N compiles but does not execute. This option creates a BATCH control file (CTL), generates an ABORT JOB message at the beginning of the log file, and returns to the RT–11 monitor.

/T:n if n=0, sets the /NOTIME option as the default on the $JOB command. If n=1, the default option on the $JOB command is /TIME.

/U indicates that the BATCH compiler must detach the BATCH run-time handler from the RT–11 monitor and unload the handler.

NOTE

You need not specify the RT–11 monitor UNLOAD BA command to remove the handler. Specifying /U to BATCH causes the handler to detach and unload.

/X indicates that the input is a precompiled BATCH program. Use this option when you do not specify the CTL file type.

RET prints the version number of the BATCH compiler.

The following example calls BATCH to compile and execute three input files (PROG1.BAT, PROG2.BAT, PROG3.BAT) to generate on DK the compiler output files, and to generate on LOG a log file:

```
.R BATCH
*PROG1.BAT,PROG2.BAT,PROG3.BAT
```

The following commands print the version number of BATCH, then compile and run SYBILD.BAT:

```
.R BATCH
* RET
BATCH V04.00A
*SYBILD
```

The following commands compile PROTO.BAT to create PROTO.CTL but do not run the compiled BATCH stream:

```
.R BATCH
*PROTO/N
```

Type the following commands to unlink BA.SYS from the monitor and to unload it:

```
.R BATCH
*/U
```

The following commands compile FILE.BAT from magtape to create FILE.CTL on RK1. They execute the compiled file and create a log file named FILE.LOG (of size 20) on LOG:

```
.R BATCH
*RK1:FILE,FILE[20]=MT:FILE
```

The following commands execute a precompiled job called FILE.TST:

```
.R BATCH
*FILE.TST/X
```

The following commands execute a precompiled job called FILE.CTL:

```
.R BATCH
*FILE/X
```

Communicating with BATCH Jobs

During the execution of a BATCH stream, BATCH can request the operator to service a peripheral device, to provide information, or to insert a command line into the BATCH stream. The operator does this by typing directives to the BATCH handler on the console terminal.

NOTE

These directives are equivalent to the compiler output that BATCH generates in the CTL file. The CTL file is an ASCII file that you can list by using the PRINT or TYPE commands or by running PIP.

These directives have the form:

\dir

where:

dir	represents one of the directives listed in Table A–6.

To use these directives, the operator must get control of the BATCH run-time handler. This can be achieved through a /WAIT or a CTY in the BATCH stream, or by typing a return on the console terminal. If a return is typed, the operator does not know exactly where the BATCH stream has been interrupted. When BATCH executes a command, it acknowledges the return and prints a return/line feed combination at the terminal. The operator can then enter a directive from Table A–6. The most useful directives are marked with an asterisk (*). Some directives are not particularly useful in this mode, but are listed to explain completely the BATCH compiler output.

Table A–6: Operator Directives to BATCH Run-Time Handler

Directive	Function
\@	Sends the characters that follow to the console terminal.
*\A	Changes the input source to be the console terminal.
*\B	Changes the input source to be the BATCH stream.
*\C	Sends the following characters to the log device.
*\D	Considers the following characters as user data.
*\E	Sends the following characters to the RT–11 monitor.
*\F	Forces the output of the current log block. If this directive is followed by any characters other than another BATCH backslash (\) directive, the BATCH job prints an error message and terminates. BATCH then returns control to the RT–11 monitor.
\G	Gets characters from the console terminal until a return is encountered.
\Hn	Help function that changes the logging mode. n specifies the following: 0 Log only .TTYOUT and .PRINT 1 Log .TTYOUT, .PRINT, and .TTYIN 2 Do not log .TTYOUT, .PRINT, and .TTYIN 3 Log only .TTYIN
\Ivxlabel1? label2? label3?	IF statement that causes conditional transfer, where v is a variable name in the range A–Z; x is a value for the signed 8-bit comparison (v-x); and label1, label2, label3 are 6-character labels to which control is transferred under certain conditions. (All labels must be six characters in length; if too short, pad with spaces.) If v-x is less than 0, control transfers to label1; if v-x is equal to 0, control goes to label2; if v-x is greater than 0, control goes to label3. The direction for the label search is indicated by ?; if ? is 0, the search begins at the beginning of this job; if ? is 1, the label search begins after the IF statement.
\Jlabel?	Jump, unconditional transfer; where label is a 6-character label and ? is 0 or 1. (All labels must be six characters in length; if too short, pad with spaces.) If ?=0, label is a backward reference; if ?=1, label is a forward reference.

Table A–6 (Cont.): Operator Directives to BATCH Run-Time Handler

Directive	Function
\Kv0	Increment variable v, where v is a variable name in the range A–Z.
\Kvln	Stores the 8-bit number n in variable v.
\Kv2	Takes the value in variable v and returns it to the program (via .TTYIN).
\Llabel	Inserts label as a 6-character alphanumeric string in the BATCH stream. (All labels must be six characters in length; if too short, pad with spaces.) Labels must not include backslash characters. Characters beyond six are ignored.

In the following example, the operator must interrupt the BATCH handler to enter information from the console. As a result of a /WAIT or 'CTY' in the BATCH stream, the following message appears at the terminal:

```
$MESSAGE/WAIT WRITE NECESSARY FILES TO DISK
```

To divert BATCH stream input from the current file to the console terminal, the operator types \E, enters commands to the RT–11 monitor, then types \B. Control then returns to the BATCH stream. The following example illustrates this procedure:

```
.R BATCH
*NEXT
 WRITE NECESSARY FILES TO DISK
?\A\E

\ECOPY DT1:FILE.MAC RK:

 FILES COPIED:
DT1:FILE.MAC   TO RK:FILE.MAC

\E\F\B

2
END BATCH
```

The following BATCH program lets you make frequent edits to a file and list only the edits. First, create a BATCH program that assembles with a listing and link the file. This BATCH program, called COMPIL.BAT, contains:

```
$JOB/RT11
   TTYIO
    !WRITE TERMINAL I/O TO LOG FILE
.R MACRO
    !CALL THE MACRO ASSEMBLER
*FILE,FILE/C=FILE
$MESSAGE/WAIT OK TO TYPE EDIT COMMANDS
.R LINK
    !CALL THE RT--11 LINKER
*FILE,LOG:=FILE
$EOJ
```

At run time, you can insert commands into the BATCH stream from the console terminal. These commands search for the section of the listing file that has been edited, then list this section to the log. You must insert the command after the

R MACRO command but before the R LINK command. The following example illustrates this procedure:

```
.R BATCH
*COMPIL
 OK TO TYPE EDIT COMMANDS
?\A\E

\ER EDIT

*ERFILE.LST$$
*EWFILE.SEC$$
*PRETRY:$=J$$
*\L$$
RETRY:  0                   ;HIGH ORDER BIT USED FOR "RESET IN PROGRESS FLAG
     49 000020  016705  177764                   MOV     RKCQE,R5     ;GET Q P
     50 000024  011502                           MOV     @R5,R2       ;R2 = BL
     51 000026  016504  000002                   MOV     2(R5),R4     ;R4 = UN
     52 000032  006204                           ASR     R4           ;ISOLATE
     53 000034  006204                           ASR     R4
     54 000036  006204                           ASR     R4
     55 000040  000304                           SWAB    R4
     56 000042  042704  017777                   BIC     #^C<160000>,R4
     57 000046  000404                           BR      2$           ;ENTER C
*EX$$

E\C\B

END BATCH
```

Terminating BATCH When BATCH terminates normally, it prints the following message and returns control to the RT–11 monitor:

```
END BATCH
```

To abort BATCH while it is executing a BATCH stream, interrupt the BATCH handler by typing a return. When BATCH executes the next command after the return, it prints a return/line feed combination at the console terminal. You then gain control of the system. Type \F followed by a return. The BATCH handler responds with the FE (forced exit) error message and writes the remainder of the log buffer. Control returns to the RT–11 monitor.

Typing two CTRL/Cs terminates BATCH immediately. Use two CTRL/Cs when BATCH is in a loop or when a long assembly is running. In these cases, BATCH responds slowly to your return interrupt.

RT–11 BATCH and RSX–11D BATCH

Some programmers run their RT–11 BATCH programs under RSX–11D. Note the differences between the two BATCH implementations listed in Table A–7. BATCH programs that run under both systems must be compatible with both RT–11 and RSX–11D BATCH.

Table A–7: Differences Between RT–11 BATCH and RSX–11D BATCH

Characteristic	RT–11	RSX–11D
File descriptors	filespec/option	SY:filnam.typ/option
Default listing file type	LST(or LIS)	LIS
Executable file type	SAV	EXE
Incompatible commands	$BASIC	
	$CALL	$MCR
	$CHAIN	
	$LIBRARY	
	$RT11	
	$SEQUENCE	
Incompatible options	$COPY/DELETE	
	$CREATE/DOLLARS	
	$CREATE/LIST	
	$DATA/DOLLARS	
	$DATA/LIST	
	$DIR file/LIST	$DIR file/DIRECTORY
	$DISMOUNT/WAIT	
	$DISMOUNT lun:/LOGICAL	
	$FORTRAN/DOLLARS	
	$FORTRAN/MAP	
	$JOB/BANNER	$JOB/NAME
	$JOB/LIST	$JOB/LIMIT
	$JOB/RT11	$JOB/MCR
	$JOB/TIME	
	$JOB/UNIQUE	
	$LINK/LIBRARY	$JOB/MCR

Table A–7 (Cont.): Differences Between RT–11 BATCH and RSX–11D BATCH

Characteristic	RT–11	RSX–11D
	$LINK/OBJECT	
	$MACRO/CREF	
	$MACRO/DOLLARS	
	$MACRO/LIBRARY	
	$MACRO/MAP	
	$MESSAGE/WAIT	
	$MESSAGE/WRITE	
	$PRINT/DELETE	
$DATA input	Appears as if from input	Appears as if from a file named FOR001.DAT
Logical device names	In $MOUNT and $DISMOUNT	Logical unit numbers only
$RUN	You must specify file name	RSX11DBAT.EXE is default

Appendix B

DCL Command and Utility Program Equivalents

This appendix provides a table of correspondence between the DCL monitor commands with their options and the system utility programs with their options. Remember that the syntax you use to issue a DCL command is different from the syntax that the Command String Interpreter requires for input and output specifications. Bear in mind that there are many differences between issuing a DCL command and running a utility program.

The following table lists all the DCL monitor commands and options. A dash under the corresponding system program or option column indicates that the command has no real system program equivalent, that the function is inherent in the keyboard monitor, or that the function is the default mode of operation.

DCL Command	**Option**	**Utility Program**	**Option**
ABORT		–	–
ASSIGN		–	–
B		–	–
BACKUP	BUP	–	–
	/DEVICE	BUP	/I
	/DIRECTORY	BUP	TT: as 3rd output specification
	/DIRECTORY/OUTPUT:filespec	BUP	3rd output specification
	/DIRECTORY/PRINTER	BUP	LP: as 1st output specification
	/FILE	BUP	/F
	/NOQUERY	BUP	/Y
	/NOREWIND	BUP	/M
	/NOSCAN	BUP	/G
	/NOLOG	BUP	/W
	/RESTORE	BUP	/X
	/SUBSET	BUP	/R
	/SAVESET	BUP	/S
	/SYSTEM	BUP	/E
	/VERIFY[:ONLY]	BUP	/V[:ONL]

DCL Command	Option	Utility Program	Option
	/INITIALIZE	BUP	/Z
BOOT		DUP	/O
	/FOREIGN	DUP	/Q
	/WAIT	DUP	/W
CLOSE		–	–
COMPILE		–	–
	/ALLOCATE:size	–	[size]
	/ALPHABETIZE	DIBOL	/A
	/BUFFERING	DIBOL	/B
	/CODE:type	FORTRAN	/I:type
	/CROSSREFERENCE[:type...]	MACRO,DIBOL	/C[:type...]
	/DIAGNOSE	FORTRAN	/B
	/DIBOL	DIBOL	–
	/DISABLE:type[:type...]	MACRO	/D:type[:type...]
	/ENABLE:type[:type...]	MACRO	/E:type[:type...]
	/EXTEND	FORTRAN	/E
	/FORTRAN	FORTRAN	–
	/HEADER	FORTRAN	/O
	/I4	FORTRAN	/T
	/LIBRARY	MACRO	/M
	/LINENUMBERS	DIBOL,FORTRAN	–
	/NOLINENUMBERS	DIBOL	/O
		FORTRAN	/S
	/LIST[:filespec]	–	2nd output specification
	/LOG	DIBOL	/G
	/MACRO	MACRO	–
	/OBJECT[:filespec]	–	1st output specification
	/NOOBJECT	–	null 1st output specification
	/ONDEBUG	DIBOL,FORTRAN	/D
	/PAGE:value	DIBOL	/P:value
	/RECORD:length	FORTRAN	/R:length
	/SHOW:type	FORTRAN,MACRO	/L:type
	/NOSHOW:type	MACRO	/N:type
	/STATISTICS	FORTRAN	/A
	/SWAP	FORTRAN	–
	/NOSWAP	FORTRAN	/U
	/TABLES	DIBOL	/S
	/UNITS:value	FORTRAN	/N:value
	/VECTORS	FORTRAN	–

DCL Command	Option	Utility Program	Option
	/NOVECTORS	FORTRAN	/V
	/WARNINGS	DIBOL	–
		FORTRAN	/W
	/NOWARNINGS	DIBOL	/W
		FORTRAN	–
COPY		PIP	–
	/ALLOCATE:size	–	[size]
	/ASCII	PIP,FILEX	/A
	/BEFORE[:date]	PIP	/J[:date]
	/BINARY	PIP	/B
	/BOOT[:dev]	DUP	/U[:dev]
	/CONCATENATE	PIP	/U
	/DATE[:date]	PIP	/C[:date]
	/DELETE	PIP	/D
	/DEVICE	DUP	/I
	/DOS	FILEX	/S
	/END:value	DUP	/E:value
	/EXCLUDE	PIP	/P
	/FILES	DUP	/F
	/IGNORE	PIP	/G:value
	/IMAGE	PIP,FILEX	/I
	/INFORMATION	PIP	/X
	/INTERCHANGE[:size]	FILEX	/U[:size]
	/LOG	PIP	/W
	/NOLOG	PIP	–
	/MULTIVOLUME	PIP	/V
	/NEWFILES	PIP	/C
	/OWNER[:valuenn,nnn]	FILEX	[UIC]
	/PACKED	FILEX	/P
	/POSITION[:value]	PIP	/M[:value]
	/PREDELETE	PIP	/O
	/PROTECTION	PIP	/F
	/NOPROTECTION	PIP	/Z
	/QUERY	PIP,FILEX	/Q
	/NOQUERY	PIP	–
	/REPLACE	DUP	/R
	/NOREPLACE	PIP	/N
	/SETDATE[:date]	PIP	/T[:date]
	/SINCE[:date]	PIP	/I[:date]
	/SLOWLY	PIP	/S
	/START:value	DUP	/G
	/SYSTEM	PIP	/Y
	/TOPS	FILEX	/T
	/VERIFY	PIP,	/H

DCL Command	Option	Utility Program	Option
		DUP	/H
	/WAIT	DUP, FILEX	/W
		PIP	/E
CREATE		DUP	/C
	/ALLOCATE:size	DUP	[size]
	/EXTENSION:size	DUP	/T:size
	/START:value	DUP	/G:value
D		–	–
DATE		–	–
DEASSIGN		–	–
DELETE		PIP	/D
	/BEFORE[:date]	PIP	/J[:date]
	/DATE[:date]	PIP	/C[:date]
	/DOS	FILEX	/S
	/ENTRY	QUEMAN	/M
	/EXCLUDE	PIP	/P
	/INFORMATION	PIP	/X
	/INTERCHANGE	FILEX	/U
	/LOG	PIP	/W
	/NEWFILES	PIP	/C
	/POSITION[:value]	PIP	/M[:value]
	/QUERY	PIP	/Q
	/NOQUERY	PIP	–
	/SINCE[:date]	PIP	/I[:date]
	/SYSTEM	PIP	/Y
	/WAIT	PIP	/E
		FILEX	/W
DIBOL		R DIBOL	–
	/ALLOCATE:size		[size]
	/ALPHABETIZE	DIBOL	/A
	/BUFFERING	DIBOL	/B
	/CROSSREFERENCE	DIBOL	/C
	/LINENUMBERS	DIBOL	–
	/NOLINENUMBERS	DIBOL	/O
	/LIST[:filespec]	DIBOL	2nd output specification
	/LOG	DIBOL	/G
	/OBJECT[:filespec]	DIBOL	1st output specification

DCL Command	Option	Utility Program	Option
	/NOOBJECT	DIBOL	null 1st output specification
	/ONDEBUG	DIBOL	/D
	/PAGE:value	DIBOL	/P:value
	/TABLES	DIBOL	/S
	/WARNINGS	DIBOL	–
	/NOWARNINGS	DIBOL	/W
DIFFERENCES	R SRCCOM		–
	/ALLOCATE:size	–	[size]
	/ALWAYS	BINCOM	/O
	/AUDITTRAIL	SRCCOM	/A
	/BINARY	BINCOM	–
	/BLANKLINES	SRCCOM	/B
	/BYTES	BINCOM	/B
	/CHANGEBAR	SRCCOM	/D
	/COMMENTS	SRCCOM	–
	/NOCOMMENTS	SRCCOM	/C
	/DEVICE	BINCOM	/D
	/END[:value]	BINCOM	/E[:value]
	/FORMFEED	SRCCOM	/F
	/MATCH[:value]	SRCCOM	/L[:value]
	/OUTPUT:filespec	SRCCOM	1st output specification
		BINCOM	1st output specification
	/PRINTER	SRCCOM	LP: as 1st output specification
		BINCOM	LP: as 1st output specification
	/QUIET	BINCOM	/Q
	/SIPP:filespec	BINCOM	2nd output specification
	/SLP:filespec	SRCCOM	2nd output specification
	/SPACES	SRCCOM	–
	/NOSPACES	SRCCOM	/S
	/START[:value]	BINCOM	/S[:value]
	/TERMINAL	SRCCOM	TT: as 1st output specification
		BINCOM	TT: as 1st output specification
	/TRIM	SRCCOM	–

DCL Command	Option	Utility Program	Option
	/NOTRIM	SRCCOM	/T
DIRECTORY	DIR		–
	/ALLOCATE:size	–	[size]
	/ALPHABETIZE	DIR	/A
	/BACKUP	BUP	/L
	/BADBLOCKS	DUP	/K
	/BEFORE[:date]	DIR	/K[:date]
	/BEGIN	DIR	/G
	/BLOCKS	DIR	/B
	/BRIEF	DIR, FILEX	/F
	/COLUMNS:value	DIR	/C:value
	/DATE[:date]	DIR	/D[:date]
	/DELETED	DIR	/Q
	/DOS	FILEX	/S
	/END:value	DUP	/E:value
	/EXCLUDE	DIR	/P
	/FAST	DIR, FILEX	/F
	/FILES	DUP	/F
	/FREE	DIR	/M
	/FULL	DIR	/E
	/INTERCHANGE	FILEX	/U
	/NEWFILES	DIR	/D
	/OCTAL	DIR	/O
	/ORDER[:category]	DIR	/S[:category]
	/OUTPUT:filespec	DIR	1st output specification
		FILEX	TT: as 1st output specification
	/OWNER[:valuenn,nnn]	FILEX	[UIC]
	/POSITION	DIR	/B
	/PRINTER	DIR	LP: as 1st output specification
		FILEX	LP: as 1st output specification
	/PROTECTION	DIR	/T
	/NOPROTECTION	DIR	/U
	/REVERSE	DIR	/R
	/SINCE[:date]	DIR	/J[:date]
	/SORT[:category]	DIR	/S[:category]
	/START:value	DUP	/G:value
	/SUMMARY	DIR	/N

DCL Command	Option	Utility Program	Option
	/TERMINAL	DI	TT: as 1st output specification
		FILEX	TT: as 1st output specification
	/TOPS	FILEX	/T
	/VOLUMEID[:ONLY]	DIR	
		FILEX	/V[:ONL]
	/WAIT	DUP,FILEX	/W
DISMOUNT		LD	/L
DUMP		R DUMP	–
	/ALLOCATE:size	–	–
	/ASCII	DUMP	–
	/NOASCII	DUMP	/N
	/BYTES	DUMP	/B
	/END:value	DUMP	/E:value
	/FOREIGN	DUMP	/T
	/IGNORE	DUMP	/G
	/ONLY:value	DUMP	/O:value
	/OUTPUT:filespec	DUMP	1st output specification
	/PRINTER	DUMP	LP: as 1st output specification
	/RAD50	DUMP	/X
	/START:value	DUMP	/S:value
	/TERMINAL	DUMP	TT: as 1st output specification
	/WORDS	DUMP	/W
E		–	–
EDIT		EDIT,TECO KED,KEX	
	/ALLOCATE:size	–	[size]
	/CREATE	EDITOR	–
	/EDIT	EDIT	–
	/EXECUTE:filespec	TECO	–
	/INSPECT	EDITOR	–
	/KED	KED	–
	/KEX	KEX	–
	/OUTPUT:filespec	EDITOR	–
	/TECO	TECO	–

DCL Command	Option	Utility Program	Option
EXECUTE		–	–
	/ALLOCATE:size	–	[size]
	/ALPHABETIZE	DIBOL	/A
	/BOTTOM:value	LINK	/B:value
	/BUFFERING	DIBOL	/B
	/CODE:type	FORTRAN	/I:type
	/CROSSREFERENCE[:type[...]	DIBOL,MACRO	/C
	/DEBUG[:filespec]	LINK	–
	/DIAGNOSE	FORTRAN	/B
	/DIBOL	DIBOL	–
	/DISABLE:type[...]	MACRO	/D
	/DUPLICATE	LINK	/D
	/ENABLE:type[...]	MACRO	/E
	/EXECUTE[:filespec]	LINK	1st output specification
	/EXTEND	FORTRAN	/E
	/FORTRAN	FORTRAN	–
	/GLOBAL	LINK	/N
	/HEADER	FORTRAN	/O
	/I4	FORTRAN	/T
	/LIBRARY	MACRO	/M
	/LINENUMBERS	DIBOL, FORTRAN	–
	/NOLINENUMBERS	DIBOL	/O
		FORTRAN	/S
	/LINKLIBRARY:filespec	LINK	–
	/LIST[:filespec]	—	2nd output specification
	/LOG	DIBOL	/G
	/MACRO	MACRO	–
	/MAP[:filespec]	LINK	2nd output specification
	/OBJECT[:filespec]	—	1st output specification
	/ONDEBUG	DIBOL, FORTRAN	/D
	/PAGE:value	DIBOL	/P:value
	/PROMPT	LINK	//
	/RECORD:length	FORTRAN	/R:length
	/RUN	RUN	–
	/NORUN	–	–
	/SHOW[:type]	FORTRAN, MACRO	/L[:type]
	/NOSHOW[:type]	MACRO	/N[:type]
	/STATISTICS	FORTRAN	/A
	/SWAP	FORTRAN	–
	/NOSWAP	FORTRAN	/U
	/TABLES	DIBOL	/S

DCL Command	Option	Utility Program	Option
	/UNITS:value	FORTRAN	/N:value
	/VECTORS	FORTRAN	–
	/NOVECTORS	FORTRAN	/V
	/WARNINGS	FORTRAN	/W
		DIBOL	–
	/NOWARNINGS	DIBOL	/W
		FORTRAN	–
	/WIDE	LINK	/W
FORMAT		–	–
	/PATTERN[:value]	FORMAT	/P[:value]
	/QUERY	FORMAT	–
	/NOQUERY	FORMAT	/Y
	/SINGLEDENSITY	FORMAT	/S
	/VERIFY[:ONLY]	FORMAT	/V[:ONLY]
	/WAIT	FORMAT	/W
FORTRAN		R FORTRAN	–
	/ALLOCATE:size	–	[size]
	/CODE:type	FORTRAN	/I:type
	/DIAGNOSE	FORTRAN	/B
	/EXTEND	FORTRAN	/E
	/HEADER	FORTRAN	/O
	/I4	FORTRAN	/T
	/LINENUMBERS	FORTRAN	–
	/NOLINENUMBERS	FORTRAN	/S
	/LIST[:filespec]	FORTRAN	2nd output specification
	/OBJECT[:filespec]	FORTRAN	1st output specification
	/NOOBJECT	FORTRAN	null 1st output specification
	/ONDEBUG	FORTRAN	/D
	/RECORD:length	FORTRAN	/R
	/SHOW[:type]	FORTRAN	/L[:type]
	/STATISTICS	FORTRAN	/A
	/SWAP	FORTRAN	–
	/NOSWAP	FORTRAN	/U
	/UNITS:value	FORTRAN	/N:value
	/VECTORS	FORTRAN	–
	/NOVECTORS	FORTRAN	/V
	/WARNINGS	FORTRAN	/W
	/NOWARNINGS	FORTRAN	–
FRUN		–	–

DCL Command	Option	Utility Program	Option
	/BUFFER:value	–	–
	/NAME:valueame	–	–
	/PAUSE	–	–
	/TERMINAL:value	–	–
GET		–	–
HELP		–	–
	/PRINTER	–	–
	/TERMINAL	–	–
INITIALIZE		DUP	/Z
	/BACKUP	BUP	/Z
	/BADBLOCKS[:RET]	DUP	/B[:RET]
	/DOS	FILEX	/S
	/FILE:filespec	DUP	1st output specification
	/INTERCHANGE	FILEX	/U
	/QUERY	DUP, FILEX	–
	/NOQUERY	BUP, DUP, FILEX	/Y
	/REPLACE[:RET]	DUP	/R[:RET]
	/RESTORE	DUP	/D
	/SEGMENTS:value	DUP	/N:value
	/VOLUMEID[:ONLY]	DUP FILEX	/V[:ONL]
	/WAIT	DUP,FILEX	/W
INSTALL		–	–
LIBRARY		R LIBR	–
	/ALLOCATE:size	–	[size]
	/CREATE	LIBR	–
	/DELETE	LIBR	/D
	/EXTRACT	LIBR	/E
	/INSERT	LIBR	–
	/LIST[:filespec]	LIBR	2nd output specification
	/MACRO[:value]	LIBR	/M[:value]
	/OBJECT[:filespec]	LIBR	1st output specification
	/NOOBJECT	LIBR	null 1st output specification
	/PROMPT	LIBR	//
	/REMOVE	LIBR	/G
	/REPLACE	LIBR	/R

DCL Command	Option	Utility Program	Option
	/UPDATE	LIBR	/U
LINK		R LINK	–
	/ALLOCATE:size	–	[size]
	/ALPHABETIZE	LINK	/A
	/BITMAP	LINK	–
	/NOBITMAP	–	/X
	/BOTTOM:value	LINK	/B:value
	/BOUNDARY:value	LINK	/Y:value
	/DEBUG[:filespec]	LINK	–
	/DUPLICATE	LINK	/D
	/EXECUTE[:filespec]	LINK	1st output specification
	/NOEXECUTE	LINK	null 1st output specification
	/EXTEND:value	LINK	/E:value
	/FILL:value	LINK	/Z:value
	/FOREGROUND[:stacksize]	LINK	/R[:stacksize]
	/GLOBAL	LINK	/N
	IDSPACE	LINK	J
	/INCLUDE	LINK	/I
	/LDA	LINK	/L
	/LIBRARY:filespec	LINK	–
	/LIMIT[:value]	LINK	/K[:value]
	/LINKLIBRARY:filespec	LINK	–
	/MAP[:filespec]	LINK	2nd output specification
	/PROMPT	LINK	//
	/ROUND:value	LINK	/U:value
	/RUN	LINK, RUN	–
	/SLOWLY	LINK	/S
	/STACK[:value]	LINK	/M[:value]
	/SYMBOLTABLE[:filespec]	LINK	3rd output specification
	/TOP[:value]	LINK	/H[:value]
	/TRANSFER[:value]	LINK	/T[:value]
	/WIDE	LINK	/W
	/XM	LINK	/V
LOAD		–	–
MACRO		R MACRO	–
	/ALLOCATE:size	–	[size]
	/CROSSREFERENCE[:type[...]]	MACRO	/C
	/DISABLE:type[...]	MACRO	/D

DCL Command	Option	Utility Program	Option
	/ENABLE:type[...]	MACRO	/E
	/LIBRARY	MACRO	/M
	/LIST[:filespec]	MACRO	2nd output specification
	/OBJECT[:filespec]	MACRO	1st output specification
	/NOOBJECT	MACRO	null 1st output specification
	/SHOW:type	MACRO	/L
	/NOSHOW:type	MACRO	/N
MOUNT		LD	/L
	/WRITE	LD	/W
	/NOWRITE	LD	/R
PRINT		–	–
	/BEFORE[:date]	PIP QUEMAN	/J[:date]
	/COPIES:value	PIP, QUEMAN	/K:value
	/DATE[:date]	PIP, QUEMAN	/C[:date]
	/DELETE	PIP, QUEMAN	/D
	/FLAGPAGE:value	QUEMAN	/H:value
	/NOFLAGPAGE	QUEMAN	/N
	/INFORMATION	PIP, QUEMAN	/X
	/LOG	PIP, QUEMAN	/W
	/NOLOG	PIP, QUEMAN	–
	/NAME:[dev:]jobname	QUEMAN	1st output specification
	/NEWFILES	PIP,QUEMAN	/C
	/PRINTER	PIP	LP: as 1st output specification
	/PROMPT	QUEMAN	//
	/QUERY	PIP,QUEMAN	/Q
	/SINCE[:date]	PIP QUEMAN	/I[:date]
	/WAIT	PIP	/E
PROTECT		PIP	/F
	/BEFORE[:date]	PIP	/J[:date]
	/DATE[:date]	PIP	/C[:date]
	/EXCLUDE	PIP	/P
	/INFORMATION	PIP	/X
	/LOG	PIP	/W
	/NOLOG	PIP	–

DCL Command	Option	Utility Program	Option
	/NEWFILES	PIP	/C
	/QUERY	PIP	/Q
	/SETDATE[:date]	PIP	/T[:date]
	/SINCE[:date]	PIP	/I[:date]
	/SYSTEM	PIP	/Y
	/WAIT	PIP	/E
R		–	–
REENTER		–	–
REMOVE		–	–
RENAME		PIP	–
	/BEFORE[:date]	PIP	/J[:date]
	/DATE[:date]	PIP	/C[:date]
	/INFORMATION	PIP	/X
	/LOG	PIP	/W
	/NOLOG	PIP	–
	/NEWFILES	PIP	/C
	/PROTECTION	PIP	/F
	/NOPROTECTION	PIP	/Z
	/QUERY	PIP	/Q
	/REPLACE	PIP	–
	/NOREPLACE	PIP	/N
	/SETDATE[:date]	PIP	/T[:date]
	/SINCE[:date]	PIP	/I[:date]
	/SYSTEM	PIP	/Y
	/WAIT	PIP	/E
RESET		–	–
RESUME		–	–
RUN		–	–
SAVE		–	–
SET		–	–
SHOW	ALL	RESORC	/A
	CONFIGURATION	RESORC	/Z
	DEVICES[:dd]	RESORC	/D[:dd]
	ERRORS	ERROUT	–
	/ALL	ERROUT	/A

DCL Command	Option	Utility Program	Option
	/FILE:filespec	ERROUT	1st input specification
	/FROM[:date]	ERROUT	/F
	/OUTPUT:filespec	ERROUT	1st output specification
	/PRINTER	ERROUT	LP: as 1st output specification
	/SUMMARY	ERROUT	/S
	/TERMINAL	ERROUT	TT: as 1st output specification
	/TO[:date]	ERROUT	/T
	JOBS	RESORC	/J
	MEMORY	RESORC	/X
	QUEUE	QUEMAN	/Q
	SUBSET	RESORC	/S
	TERMINALS	RESORC	/T
SQUEEZE		DUP	/S
	/OUTPUT:filespec	DUP	1st output specification
	/QUERY	DUP	–
	/NOQUERY	DUP	/Y
	/WAIT	DUP	/W
SRUN			
	/BUFFER:value	–	–
	/LEVEL:value	–	–
	/NAME:logical–jobname	–	–
	/PAUSE	–	–
	/TERMINAL:value	–	–
START		–	–
SUSPEND		–	–
TIME		–	–
TYPE		–	–
	/BEFORE[:date]	PIP	/J[:date]
	/COPIES:value	PIP	/K:value
	/DATE[:date]	PIP	/C[:date]
	/DELETE	PIP	/D
	/INFORMATION	PIP	/X
	/LOG	PIP	/W
	/NOLOG	PIP	–

DCL Command	Option	Utility Program	Option
	/NEWFILES	PIP	/C
	/QUERY	PIP	/Q
	/SINCE[:date]	PIP	/I[:date]
	/WAIT	PIP	/E
UNLOAD		–	–
UNPROTECT		PIP	/F
	/BEFORE[:date]	PIP	/J[:date]
	/DATE[:date]	PIP	/C[:date]
	/EXCLUDE	PIP	/P
	/INFORMATION	PIP	/X
	/LOG	PIP	/W
	/NOLOG	PIP	–
	/NEWFILES	PIP	/C
	/QUERY	PIP	/Q
	/SETDATE[:date]	PIP	/T[:date]
	/SINCE[:date]	PIP	/I[:date]
	/SYSTEM	PIP	/Y
	/WAIT	PIP	/E

Index