VMS DCL Dictionary : Part I

Order Number: AA–PBK5A–TE

June 1990

This manual provides detailed reference information and examples for all VMS DCL commands and lexical functions.

Revision/Update Information: This manual supersedes the *VMS DCL Dictionary*, Version 5.3.

Software Version: VMS Version 5.4

The postpaid Reader's Comments forms at the end of this document request your critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

CDA	DEQNA	MicroVAX	VAX RMS
DDIF	Desktop–VMS	PrintServer 40	VAXserver
DEC	DIGITAL	Q-bus	VAXstation
DECdtm	GIGI	ReGIS	VMS
DECnet	HSC	ULTRIX	VT
DECUS	LiveLink	UNIBUS	XUI
DECwindows	LN03	VAX	
DECwriter	MASSBUS	VAXcluster	digital™

The following are third-party trademarks:

Adobe, Display PostScript, and PostScript are registered trademarks of Adobe Systems Incorporated.

ZK9996

Production Note

This book was produced with the VAX DOCUMENT electronic publishing system, a software tool developed and sold by Digital. In this system, writers use an ASCII text editor to create source files containing text and English-like code; this code labels the structural elements of the document, such as chapters, paragraphs, and tables. The VAX DOCUMENT software, which runs on the VMS operating system, interprets the code to format the text, generate a table of contents and index, and paginate the entire document. Writers can print the document on the terminal or line printer, or they can use Digital-supported devices, such as the LN03 laser printer and PostScript printers (PrintServer 40 or LN03R ScriptPrinter), to produce a typeset-quality copy containing integrated graphics.

Contents

PART I

Contents

Contents

INDEX

PART II

Contents

INDEX

FIGURES

TABLES

Contents

Preface

Intended Audience

This manual is intended for all users of the VMS operating system. It includes descriptions of all Digital Command Language (DCL) commands and lexical functions. If a command has any restrictions or requires special privileges, they are noted in reference information for that command.

Readers of this manual should be familiar with the material covered in the *VMS DCL Concepts Manual*. Furthermore, while familiarity with the *Guide to Using VMS Command Procedures* is not a requirement for using this manual, it does help clarify some of the examples involving command procedures.

Document Structure

This manual contains detailed descriptions of each command and lexical function. The commands are listed in alphabetical order, with the command name appearing at the top of every page. The lexical functions are grouped under the heading "Lexical Functions" (after the JOB command description) and are listed alphabetically within that grouping; the lexical function name appears at the top of each page.

The *VMS DCL Dictionary* is a two-part manual. Part I contains commands beginning with the letters A to M (including the lexical functions); Part II contains commands beginning with the letters N to Z. The Table of Contents and Index are comprehensive: they include both parts.

The commands that invoke language compilers and other VAX optional software products are not included in this manual; they are included in the documentation provided with those products.

Associated Documents

This manual is Part I of a two-part manual; it contains DCL commands beginning with the letters A to M, as well as the lexical functions. For the remaining commands, see Part II.

For an introduction to the VMS operating system and for information on using DCL, see the *Introduction to VMS*. This manual is especially recommended for novice users or users lacking experience with interactive computer systems.

The *VMS DCL Concepts Manual* provides an overview of DCL command language concepts.

The *Guide to Using VMS Command Procedures* defines and illustrates good practices in constructing command procedures with DCL commands and lexical functions.

Preface

The various VMS utilities reference manuals document major VMS utilities. These manuals describe the DCL commands that invoke the various utilities, describe any commands that you can enter while running a utility, and provide reference information. For all utilities documented in these volumes, the *VMS DCL Dictionary* provides only a brief description and format information.

The *VMS System Messages and Recovery Procedures Reference Manual* explains what the messages mean and, where applicable, suggests actions for you to take.

The *Overview of VMS Documentation* describes the new organization of the VMS document set. This manual shows how the individual manuals fit together and relate to each other.

Conventions

The following conventions are used in this manual:

mouse	The term *mouse* is used to refer to any pointing device, such as a mouse, a puck, or a stylus.
MB1, MB2, MB3	MB1 indicates the left mouse button, MB2 indicates the middle mouse button, and MB3 indicates the right mouse button. (The buttons can be redefined by the user.)
Ctrl/x	A sequence such as Ctrl/x indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 x	A sequence such as PF1 x indicates that you must first press and release the key labeled PF1, then press and release another key or a pointing device button.
Return	A key name is shown enclosed to indicate that you press a key on the keyboard.
. . .	In examples, a horizontal ellipsis indicates one of the following possibilities:
	• Additional optional arguments in a statement have been omitted.
	• The preceding item or items can be repeated one or more times.
	• Additional parameters, values, or other information can be entered.
. . .	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
()	In format descriptions, parentheses indicate that, if you choose more than one option, you must enclose the choices in parentheses.

[]	In format descriptions, brackets indicate that whatever is enclosed is optional; you can select none, one, or all of the choices.
{}	In format descriptions, braces surround a required choice of options; you must choose one of the options listed.
red ink	Red ink indicates information that you must enter from the keyboard or a screen object that you must choose or click on. For online versions, user input is shown in **bold**.
boldface text	Boldface text represents the introduction of a new term or the name of an argument, an attribute, or a reason.
italic text	Italic text represents information that can vary in system messages (for example, Internal error *number*).
UPPERCASE TEXT	Uppercase letters indicate that you must enter a command (for example, enter OPEN/READ).
UPPERCASE TEXT	Uppercase letters indicate the name of a routine, the name of a file, the name of a file protection code, or the abbreviation for a system privilege.
-	Hyphens in coding examples indicate that additional arguments to the request are provided on the line that follows.
numbers	Unless otherwise noted, all numbers in the text are assumed to be decimal. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.

= (Assignment Statement)

Defines a symbolic name for a character string or integer value.

FORMAT *symbol-name =[=] expression*

*symbol-name[bit-position,size] =[=]
replacement-expression*

PARAMETERS ***symbol-name***
Specifies a string of 1 to 255 characters for the symbol name. The name can contain any alphanumeric characters from the DEC Multinational Character Set, the underscore (_), and the dollar sign ($). However, the name must begin *only* with an alphabetic character (uppercase and lowercase characters are equivalent), an underscore, or a dollar sign. Using one equal sign (=) places the symbol name in the local symbol table for the current command level. Using two equal signs (==) places the symbol name in the global symbol table.

expression
Names the value on the right-hand side of an assignment statement. This parameter can consist of a character string, an integer, a symbol name, a lexical function, or a combination of these entities. The components of the expression are evaluated, and the result is assigned to the symbol. All literal character strings must be enclosed in quotation marks (" "). If the expression contains a symbol, the expression is evaluated using the symbol's value.

The result of expression evaluation is either a character string or a signed integer value. If the expression is evaluated as a string, the symbol is assigned a string value. If the expression is evaluated as an integer, the symbol is assigned an integer value. If the integer value exceeds the capacity of the 4-byte buffer that holds it, no error message is issued.

For a summary of operators used in expressions, details on how to specify expressions, and details on how expressions are evaluated, see the *VMS DCL Concepts Manual*.

DCL uses a buffer that is 1024 bytes long to hold an assignment statement and to evaluate the expression. The length of the symbol name, the expression, and the expression's calculations cannot exceed 1024 bytes.

[bit-position,size]
States that a binary overlay is to be inserted in the current 32-bit value of a symbol name. The current value of the symbol name is evaluated. Then, the specified number of bits is replaced by the result of the replacement expression. The bit position is the location relative to bit 0 at which the overlay is to occur. If the symbol you are overlaying is an integer, then the bit position must be less than 32. The sum of the bit position and the size must be less than or equal to 32.

= (Assignment Statement)

If the symbol you are overlaying is a string, then the bit position must be less than 6152. Because each character is represented using 8 bits, you can begin an overlay at any character through the 768th character. (The 768th character starts in bit position 6144.) The sum of the bit position and the size must be less than or equal to 6152.

The size is the number of bits to be overlaid. If you specify a size that is greater than 32, DCL reduces the size to 32.

The brackets are required notation; no spaces are allowed between the symbol name and the left bracket. Specify values for the bit position and size as integers.

replacement-expression
Specifies the value that is used to overlay the symbol you are modifying. Specify the replacement expression as an integer.

If the symbol you are modifying is an integer, the replacement expression defines a bit pattern that is overlaid on the value assigned to the symbol. If the symbol you are modifying is a character string, the result of the replacement expression defines a bit pattern that is overlaid on the specified bits of the character string. If the symbol you are modifying is undefined, the result of the replacement expression is overlaid on a null string.

DESCRIPTION Symbols defined using assignment statements allow you to extend the command language. At the interactive command level, you can use symbols to define synonyms for commands or command lines. In command procedure files, you can use symbols to provide for conditional execution and substitution of variables.

The maximum number of symbols that can be defined at any time depends on the following:

- The amount of space available to the command interpreter to contain symbol tables and labels for the current process. The amount of space is determined for each process by the SYSGEN parameter CLISYMTBL.

- The size of the symbol names and their values. The command interpreter allocates space for a symbol name and its value. In addition, a few bytes of overhead are allocated for each symbol.

EXAMPLES

1 `$ LIST == "DIRECTORY"`

The assignment statement in this example assigns the user-defined synonym LIST as a global symbol definition for the DCL command DIRECTORY.

2
```
$ COUNT = 0
$ LOOP:
$       COUNT = COUNT + 1
$       IF P'COUNT' .EQS. "" THEN EXIT
$       APPEND/NEW &P'COUNT' SAVE.ALL
$       DELETE &P'COUNT';*
$       IF COUNT .LT. 8 THEN GOTO LOOP
$ EXIT
```

This command procedure, COPYDEL.COM, appends files (specified as parameters) to a file called SAVE.ALL. After a file has been appended, the command procedure deletes the file. Up to eight file names can be passed to the command procedure. The file names are assigned to the symbols P1, P2, and so on.

The command procedure uses a counter to refer to parameters that are passed to it. Each time through the loop, the procedure uses an IF command to check whether the value of the current parameter is a null string. When the IF command is scanned, the current value of the symbol COUNT is concatenated with the letter P. The first time through the loop, the IF command tests P1; the second time through the loop it tests P2, and so on. After the expression P'COUNT' is evaluated, the substitution of the file names that correspond to P1, P2, and so on is automatic within the context of the IF command.

The APPEND and DELETE commands do not perform any substitution automatically, because they expect and require file specifications as input parameters. The ampersand (&) precedes the P'COUNT' expression for these commands to force the appropriate symbol substitution. When these commands are initially scanned each time through the loop, COUNT is substituted with its current value. Then, when the commands execute, the ampersand causes another substitution: the first file specification is substituted for P1, the second file specification is substituted for P2, and so on.

To invoke this procedure, use the following command:

```
$ @COPYDEL ALPHA.TXT BETA.DOC
```

The files ALPHA.TXT and BETA.DOC are each appended to the file SAVE.ALL and are then deleted.

3
```
$ A = 25
$ CODE = 4 + F$INTEGER("6") - A
$ SHOW SYMBOL CODE
  CODE = -15   HEX = FFFFFFF1   Octal = 1777761
```

This example contains two assignment statements. The first assignment statement assigns the value 25 to the symbol A. The second assignment statement evaluates an expression containing an integer (4), a lexical function (F$INTEGER("6")), and the symbol A. The result of the expression, −15, is assigned to the symbol CODE.

= (Assignment Statement)

4
```
$  FILENAME = "JOBSEARCH" - "JOB"
$  FILETYPE = ".OBJ"
$  FILESPEC = FILENAME + FILETYPE
$  TYPE 'FILESPEC'
```

> The first command in this example assigns the symbol FILENAME the
> value "SEARCH". Notice that the string "SEARCH" is the result of the
> string reduction operation performed by the expression. The second
> command assigns the symbol FILETYPE the character string ".OBJ".
> The symbols FILENAME and FILETYPE are then added together in
> an expression assigned to the symbol FILESPEC. Because the values of
> the symbols FILENAME and FILETYPE are concatenated, the resultant
> value assigned to FILESPEC is the character string "SEARCH.OBJ". The
> symbol FILESPEC is then used as a parameter for the TYPE command.
> The single quotation marks (' ') request the command interpreter to
> replace the symbol FILESPEC with its value SEARCH.OBJ. Thus, the
> TYPE command types the file named SEARCH.OBJ.

5
```
$  BELL[0,32] = %X07
$  SHOW SYMBOL BELL
   BELL = ""
```

> In this example, the symbol BELL is created with an arithmetic overlay
> assignment statement. Because the symbol BELL is previously undefined,
> the hexadecimal value 7 is inserted over a null character string and is
> interpreted as the ASCII code for the bell character on a terminal. When
> you issue the command SHOW SYMBOL BELL, the terminal beeps.

> If the symbol BELL had been previously defined with an integer value, the
> result of displaying BELL would have been to show its new integer value.

:= (String Assignment)

Defines a symbolic name for a character string value.

FORMAT　　　*symbol-name :=[=] string*

　　　　　　　　symbol-name[offset,size] :=[=] replacement-string

PARAMETERS　　***symbol-name***
Specifies a string of 1 to 255 characters for the symbol name. The name can contain any alphanumeric characters from the DEC Multinational Character Set, the underscore (_), and the dollar sign ($). However, the name must begin *only* with an alphabetic character, an underscore, or a dollar sign. Using one equal sign (:=) places the symbol name in the local symbol table for the current command level. Using two equal signs (:= =) places the symbol name in the global symbol table.

string
Names the character string value to be equated to the symbol. The string can contain any alphanumeric or special characters. DCL uses a buffer that is 1024 bytes long to hold a string assignment statement. Therefore, the length of the symbol name, the string, and any symbol substitution within the string cannot exceed 1024 characters.

With the string assignment statement (:=), you do not need to enclose a string literal in quotation marks (" "). String values are converted to uppercase automatically. Also, any leading and trailing spaces and tabs are removed, and multiple spaces and tabs between characters are compressed to a single space.

It is easier to use the assignment statement (=) to create symbols with string values because the assignment statement does not automatically convert letters to uppercase and remove extra spaces. Also, the assignment statement allows you to perform string operations in expressions.

To prohibit uppercase conversion and to retain required space and tab characters in a string, place quotation marks around the string. To use quotation marks in a string, enclose the entire string within quotation marks and use a double set of quotation marks within the string. For example:

```
$  TEST := "this     is a ""test"" string"
$  SHOW SYMBOL TEST
   TEST = "this     is a "test" string"
```

In this example, the spaces, lowercase letters, and quotation marks are preserved in the symbol definition.

To continue a symbol assignment on more than one line, use the hyphen (-) as a continuation character. For example:

```
$  LONG_STRING := THIS_IS_A_VERY_LONG-
_$ _SYMBOL_STRING
```

:= (String Assignment)

To assign a null string to a symbol by using the string assignment statement, do not specify a string. For example:

```
$ NULL :=
```

Specify the string as a string literal, or as a symbol or lexical function that evaluates to a string literal. If you use symbols or lexical functions, place single quotation marks (' ') around them to request symbol substitution. See the *VMS DCL Concepts Manual* for more information on symbol substitution.

You can also use the string assignment statement to define a foreign command. See the *VMS DCL Concepts Manual* for more information about foreign commands.

[offset,size]

Specifies that a portion of a symbol value is to be overlaid with a replacement string. This form of the string assignment statement evaluates the value assigned to a symbol and then replaces the portion of the value (defined by the offset and size) with the replacement string. The brackets are required notation, and no spaces are allowed between the symbol name and the left bracket.

The offset specifies the character position relative to the beginning of the symbol name's string value at which replacement is to begin. Offset values start at 0.

If the offset is greater than the offset of the last character in the string you are modifying, spaces are inserted between the end of the string and the offset where the replacement string is added. The maximum offset value you can specify is 768.

The size specifies the number of characters to replace. Size values start at 1.

Specify the offset and size as integer expressions. See the *VMS DCL Concepts Manual* for more information on integer expressions. The value of the size plus the offset must not exceed 769.

replacement-string

Specifies the string that is used to overwrite the string you are modifying. If the replacement string is shorter than the size argument, the replacement string is filled with blanks on the right until it equals the specified size. Then the replacement string overwrites the string assigned to the symbol name. If the replacement string is longer than the size argument, then the replacement string is truncated on the right to the specified size.

You can specify the replacement string as a string literal, or as a symbol or lexical function that evaluates to a string literal. If you use symbols or lexical functions, place single quotation marks (' ') around them to request symbol substitution. For more information on symbol substitution, see the *VMS DCL Concepts Manual*.

EXAMPLES

1
```
$  TIME := SHOW TIME
$  TIME
   19-APR-1990 11:55:44
```

> In this example, the symbol TIME is equated to the command string SHOW TIME. Because the symbol name appears as the first word in a command string, the command interpreter automatically substitutes it with its string value and executes the command SHOW TIME.

2
```
$  STAT := $DBA1:[CRAMER]STAT
$  STAT
```

> This example shows how to define STAT as a foreign command. The symbol STAT is equated to a string that begins with a dollar sign followed by a file specification. The command interpreter assumes that the file specification is that of an executable image, that is, a file with a file type of EXE. The symbol STAT in this example becomes a synonym for the following command:

```
$  RUN DBA1:[CRAMER]STAT.EXE
```

> When you subsequently type STAT, the command interpreter executes the image.

3
```
$  A = "this is a big     space."
$  SHOW SYMBOL A
   A = "this is a big     space."
$  B := 'A'
$  SHOW SYMBOL B
   B = "THIS IS A BIG SPACE."
```

> This example compares the assignment and the string assignment statements. The symbol A is defined using the assignment statement, so lowercase letters and multiple spaces are retained. The symbol B is defined using the string assignment statement. Note that the single quotation marks (' ') are required; otherwise, the symbol name B would have been equated to the literal string A. However, when symbol A's value is assigned to symbol B, the letters are converted to uppercase and multiple spaces are compressed.

4
```
$  FILE_NAME := MYFILE
$  FILE_NAME[0,2]:= OL
$  SHOW SYMBOL FILE_NAME
   FILE_NAME = "OLFILE"
```

> In this example, the substring expression in the assignment statement overlays the first 2 characters of the string assigned to the symbol FILE_ NAME with the letters OL. The offset of 0 requests that the overlay begin with the first character in the string, and the size specification of 2 indicates the number of characters to overlay.

:= (String Assignment)

```
5  $  FILE_NAME := MYFILE
   $  FILE_TYPE := .TST
   $  FILE_NAME[F$LENGTH(FILE_NAME),4] := 'FILE_TYPE'
   $  SHOW SYMBOL FILE_NAME
     FILE_NAME = "MYFILE.TST"
```

In this example, the symbol name FILE_NAME is equated to the string MYFILE and the symbol name FILE_TYPE is equated to the string .TST. The third assignment statement uses the lexical function F$LENGTH to define the offset value where the overlay is to begin. The symbol name FILE_TYPE is used to refer to the replacement string (.TST). Note that you must use single quotation marks ('') to request symbol substitution.

The F$LENGTH lexical function returns the length of the string equated to the symbol FILE_NAME; this length is used as the offset. The expression requests that 4 characters of the string currently equated to the symbol FILE_TYPE be placed at the end of the string currently equated to FILE_NAME. The resultant value of the symbol FILE_NAME is MYFILE.TST.

@ (Execute Procedure)

Executes a command procedure or requests the command interpreter to read subsequent command input from a specific file or device.

FORMAT

@   *filespec [parameter[,...]]*

PARAMETERS

filespec
Specifies either the input device or the file for the preceding command, or the command procedure to be executed. The default file type is COM. Wildcard characters are not allowed in the file specification.

parameter[,...]
Specifies from one to eight optional parameters to pass to the command procedure. The symbols (P1, P2, . . . P8) are assigned character string values in the order of entry. The symbols are local to the specified command procedure. Separate each parameter with one or more blanks. Use two consecutive quotation marks ("") to specify a null parameter. You can specify a parameter with a character string value containing alphanumeric or special characters, with the following restrictions:

- The command interpreter converts alphabetic characters to uppercase and uses blanks to delimit each parameter. To pass a parameter that contains embedded blanks or literal lowercase letters, place the parameter in quotation marks.

- If the first parameter begins with a slash (/), you must enclose the parameter in quotation marks (" ").

- To pass a parameter that contains literal quotation marks and spaces, enclose the entire string in quotation marks and use two consecutive quotation marks within the string. For example, the command procedure TEST.COM contains the following line:

```
$ WRITE SYS$OUTPUT P1
```

Enter the following at the DCL prompt ($):

```
$ @TEST "Never say ""quit"""
```

When the procedure TEST.COM executes, the parameter P1 is equated to the following string:

```
Never say "quit"
```

If a string contains quotation marks and does not contain spaces, the quotation marks are preserved in the string and the letters within the quotation marks remain in lowercase. For example, enter the following at the DCL prompt:

```
$ @TEST abc"def"ghi
```

@ (Execute Procedure)

When the procedure TEST.COM executes, the parameter P1 is equated to the following string:

ABC"def"GHI

To use a symbol as a parameter, enclose the symbol in single quotation marks (' ') to force symbol substitution. For example:

```
$  NAME = "JOHNSON"
$  @INFO 'NAME'
```

The single quotation marks cause the value "JOHNSON" to be substituted for the symbol NAME. Therefore, the parameter "JOHNSON" is passed as P1 to INFO.COM.

DESCRIPTION

Use the @ command to execute a command procedure that contains the following:

- DCL command lines or data, or both

- Qualifiers or parameters, or both, for a specific command line

To execute a command procedure containing commands or data, or both, place the @ command at the beginning of a command line and then specify the name of the command procedure file. The command procedure can contain DCL commands and input data for a command or program that is currently executing. All DCL commands in a command procedure must begin with a dollar sign ($). If a command is continued with a hyphen (-), the subsequent lines must not begin with a dollar sign.

Any line in a command procedure that does not contain a dollar sign in the first character position (and is not a continuation line) is treated as input data for the command or program that is currently executing. The DECK command allows you to specify that data contains dollar signs in record position one.

A command procedure can also contain the @ command to execute another command procedure. The maximum command level you can achieve by nesting command procedures is 16, including the top-level command procedure. Command procedures can also be queued for processing as batch jobs, either by using the SUBMIT command or by placing a deck of cards containing the command procedure in the system card reader.

To execute a command procedure that contains qualifiers or parameters, or both, for a specific command line, place the @ command where the qualifiers or parameters normally would be in the command line. Then specify the name of the command procedure file containing the qualifiers or parameters.

If the command procedure file begins with parameters for the command, the @ command must be preceded by a space. For example:

```
$  CREATE TEST.COM
TIME
Ctrl/Z
$  SHOW @TEST
   19-APR-1990 17:20:26
```

If the file begins with qualifiers for the command, do *not* precede the @ command with a space. For example:

```
$  CREATE TEST_2.COM
/SIZE
Ctrl/Z
$  DIR@TEST_2

Directory WORK$:[SCHEDULE]

JANUARY.TXT;8          19-APR-1990 15:47:45.57
FEBRUARY.TXT;7         19-APR-1990 15:43:16.20
MARCH.TXT;6            19-APR-1990 11:11:45.74

Total of 3 files.
```

If the file contains parameters or qualifiers, or both, do *not* begin the lines in the file with dollar signs. Any additional data on the command line following @filespec is treated as parameters for the procedure.

QUALIFIER

/OUTPUT=filespec

Specifies the name of the file to which the command procedure output is written. By default, the output is written to the current SYS$OUTPUT device. The default output file type is LIS. Wildcard characters are not allowed in the output file specification. System responses and error messages are written to SYS$COMMAND as well as to the specified file. The /OUTPUT qualifier must immediately follow the file specification of the command procedure; otherwise, the qualifier is interpreted as a parameter to pass to the command procedure.

You can also redefine SYS$OUTPUT to redirect the output from a command procedure. If you place the following command as the first line in a command procedure, output will be directed to the file you specify:

```
$ DEFINE SYS$OUTPUT filespec
```

When the procedure exits, SYS$OUTPUT will be restored to its original equivalence string. This produces the same result as using the /OUTPUT qualifier when you execute the command procedure.

EXAMPLES

1
```
$  CREATE DOFOR.COM
$ ON WARNING THEN EXIT
$ IF P1.EQS."" THEN INQUIRE P1 FILE
$ FORTRAN/LIST 'P1'
$ LINK 'P1'
$ RUN 'P1'
$ PRINT 'P1'
Ctrl/Z
$  @DOFOR AVERAGE
```

This example shows a command procedure, named DOFOR.COM, that executes the FORTRAN, LINK, and RUN commands to compile, link, and execute a program. The ON command requests that the procedure not continue if any of the commands result in warnings or errors.

@ (Execute Procedure)

When you execute DOFOR.COM, you can pass the file specification of the FORTRAN program as the parameter P1. If you do not specify a value for P1 when you execute the procedure, the INQUIRE command issues a prompting message to the terminal and equates what you enter with the symbol P1. In this example, the file name AVERAGE is assigned to P1. The file type is not included because the commands FORTRAN, LINK, RUN, and PRINT provide default file types.

2 `$ @MASTER/OUTPUT=MASTER.LOG`

This command executes a procedure named MASTER.COM; all output is written to the file MASTER.LOG.

3
```
$ CREATE FILES.COM
*.FOR, *.OBJ
Ctrl/Z
$ DIRECTORY @FILES
```

This example shows a command procedure, FILES.COM, that contains parameters for a DCL command line. You can execute this procedure after the DIRECTORY command to get a listing of all FORTRAN source and object files in your current default directory.

4
```
$ CREATE QUALIFIERS.COM
/DEBUG/SYMBOL_TABLE/MAP/FULL/CROSS_REFERENCE
Ctrl/Z
$ LINK SYNAPSE@QUALIFIERS
```

This example shows a command procedure, QUALIFIERS.COM, that contains qualifiers for the LINK command. When you enter the LINK command, specify the command procedure immediately after the file specification of the file you are linking. Do not type a space between the file specification and the @ command.

5
```
$ CREATE SUBPROCES.COM
$ RUN 'P1' -
  /BUFFER_LIMIT=1024 -
  /FILE_LIMIT=4 -
  /PAGE_FILES=256 -
  /QUEUE_LIMIT=2 -
  /SUBPROCESS_LIMIT=2 -
  'P2'  'P3'  'P4'  'P5'  'P6'  'P7'  'P8'
Ctrl/Z
$ @SUBPROCES  LIBRA  /PROCESS_NAME=LIBRA
```

This example shows a command procedure named SUBPROCES.COM. This procedure issues the RUN command to create a subprocess to execute an image and also contains qualifiers defining quotas for subprocess creation. The name of the image to be run is passed as the parameter P1. Parameters P2 to P8 can be used to specify additional qualifiers.

In this example, the file name LIBRA is equated to P1; it is the name of an image to execute in the subprocess. The qualifier /PROCESS_NAME=LIBRA is equated to P2; it is an additional qualifier for the RUN command.

```
6   $  CREATE EDOC.COM
    $ ASSIGN SYS$COMMAND:   SYS$INPUT
    $ NEXT:
    $      INQUIRE NAME "File name"
    $      IF NAME.EQS."" THEN EXIT
    $      EDIT/EDT 'NAME'.DOC
    $      GOTO NEXT
    Ctrl/Z
    $  @EDOC
```

This procedure, named EDOC.COM, invokes the EDT editor. When an edit session is terminated, the procedure loops to the label NEXT. Each time through the loop, the procedure requests another file name for the editor and supplies the default file type of DOC. When a null line is entered in response to the INQUIRE command, the procedure terminates with the EXIT command.

The ASSIGN command changes the equivalence name of SYS$INPUT for the duration of the procedure. This change allows the EDT editor to read input data from the terminal, rather than from the command procedure file (the default input data stream if SYS$INPUT had not been changed). When the command procedure exits, SYS$INPUT is reassigned to its original value.

ACCOUNTING

Invokes the Accounting Utility, which reports accounting data. For a complete description of the Accounting Utility, see the *VMS Accounting Utility Manual*.

FORMAT   **ACCOUNTING**   *[filespec[,...]]*

ALLOCATE

Provides your process with exclusive access to a device until you deallocate the device or terminate your process. Optionally associates a logical name with the device.

FORMAT   **ALLOCATE**   *device-name[:][,...] [logical-name[:]]*

PARAMETERS   ***device-name[:][,...]***
Specifies the name of a physical device or a logical name that translates to the name of a physical device. The device name can be generic: if no controller or unit number is specified, any device that satisfies the specified part of the name is allocated. If more than one device is specified, the first available device is allocated.

logical-name[:]
Specifies a string of 1 to 255 alphanumeric characters. Enclose the string in single quotation marks (' ') if it contains blanks. Trailing colons (:) are not used. The name becomes a process logical name with the device name as the equivalence name. The logical name remains defined until it is explicitly deleted or your process terminates.

QUALIFIERS   ***/GENERIC***
/NOGENERIC (default)
Indicates that the first parameter is a device *type* rather than a device *name*. Example device types are: RX50, RD52, TK50, RC25, RCF25, and RL02. The first free, nonallocated device of the specified name and type is allocated.

The /[NO]GENERIC qualifier is placed before the *device-name* parameter in the ALLOCATE command line. For example, you can allocate an RK07 device by entering the following command at the DCL prompt ($):

```
$  ALLOCATE/GENERIC RK07 DISK
```

The following table shows some device types that you can specify with the /GENERIC qualifier:

Disk Devices	Tape Devices
RA60/70/80/81/90	TA78/79/81
RC25/RCF25	TK50/70
RK06/7	TS11
RL01/2	TU16
RM03/05/80	TU58
RP04/5/6/7	TU77/78/79/80/81

ALLOCATE

Disk Devices	Tape Devices
RX01/2/4/33	
RZ55	

/LOG (default)
/NOLOG

Displays a message indicating the name of the device allocated. If the operation specifies a logical name that is currently assigned to another device, then the superseded value is displayed.

EXAMPLES

1
```
$ ALLOCATE  DMB2:
%DCL-I-ALLOC, _DMB2: allocated
```

The ALLOCATE command in this example requests the allocation of a specific RK06/RK07 disk drive, that is, unit 2 on controller B. The system response indicates that the device was allocated successfully.

2
```
$ ALLOCATE  MT,MF:   TAPE:
%DCL-I-ALLOC, _MTB2: allocated
   .
   .
   .
$ SHOW LOGICAL TAPE:
TAPE: = _MTB2:     (process)
$ DEALLOCATE TAPE:
$ DEASSIGN TAPE:
```

The ALLOCATE command in this example requests the allocation of a tape device whose name begins with MT or MF and assigns it the logical name TAPE. The ALLOCATE command locates an available tape device whose name begins with MT, and responds with the name of the device allocated. (If no tape device beginning with MT had been found, the ALLOCATE command would have searched for a device beginning with MF.) Subsequent references to the device TAPE in user programs or command strings are translated to the device name MTB2.

When the tape device is no longer needed, the DEALLOCATE command deallocates it and the DEASSIGN command deletes the logical name. Note that the logical name TAPE was specified with a colon on the ALLOCATE command, but that the logical name table entry does not have a colon.

3
```
$ ALLOCATE/GENERIC RL02 WORK
%DCL-I-ALLOC, _DLA1: allocated
%DCL-I-SUPERSEDE, previous value of WORK has been superseded
```

The ALLOCATE command in this example requests the allocation of any RL02 disk device and assigns the logical name WORK to the device. The completion message identifies the allocated device and indicates that the assignment of the logical name WORK supersedes a previous assignment of that name.

4   `$ ALLOCATE $TAPE1`
`%DCL-I-ALLOC, _MUA0: allocated`

> The ALLOCATE command in this example allocates the tape device MUA0, which is associated with the logical name $TAPE1.

5   `$ ALLOCATE /GENERIC RX50 ACCOUNTS`

> The ALLOCATE command in this example allocates the first free floppy disk drive and makes its name equivalent to the process logical name ACCOUNTS.

ANALYZE/AUDIT

Invokes the Audit Analysis Utility, which selectively extracts and displays information from security audit log files or security archive files. For a complete description of the Audit Analysis Utility, see the *VMS Audit Analysis Utility Manual*.

FORMAT   **ANALYZE/AUDIT**   *[filespec]*

ANALYZE/CRASH_DUMP

Invokes the System Dump Analyzer Utility, which analyzes a system dump file. The /CRASH_DUMP qualifier is required. For a complete description of the System Dump Analyzer Utility, see the *VMS System Dump Analyzer Utility Manual*.

FORMAT **ANALYZE/CRASH_DUMP** *filespec*

ANALYZE/DISK_STRUCTURE

Invokes the Analyze/Disk_Structure Utility, which does the following:

- Checks the readability and validity of Files–11 On-Disk Structure Level 1 and Files–11 On-Disk Structure Level 2 disk volumes.

- Reports errors and inconsistencies.

The /DISK_STRUCTURE qualifier is required. For a complete description of the Analyze/Disk_Structure Utility, see the *VMS Analyze/Disk_Structure Utility Manual*.

FORMAT   **ANALYZE/DISK_STRUCTURE** *device-name[:]*

ANALYZE/ERROR_LOG

Invokes the Errorlog Report Formatter, which reports selectively the contents of an error log file. The /ERROR_LOG qualifier is required. For a complete description of the Error Log Utility, see the *VMS Error Log Utility Manual.*

FORMAT   **ANALYZE/ERROR_LOG**   *[filespec[,...]]*

ANALYZE/IMAGE

Analyzes the contents of an executable image file or a shareable image file and checks for obvious errors in the image file. The /IMAGE qualifier is required. For general information about image files, see the description of the linker in the *VMS Linker Utility Manual*. (Use the ANALYZE/OBJECT command to analyze the contents of an object file.)

FORMAT   **ANALYZE/IMAGE** *filespec[,...]*

PARAMETER   *filespec[,...]*

Specifies the name of one or more image files that you want analyzed. You must specify at least one file name. If you specify more than one file, separate the file specifications with either commas (,) or plus signs (+). The default file type is EXE.

Wildcard characters (* and %) are allowed in the file specification.

DESCRIPTION

The ANALYZE/IMAGE command provides a description of the components of an executable image file or shareable image file. It also verifies that the structure of the major parts of the image file is correct. However, the ANALYZE/IMAGE command cannot ensure that program execution is error free.

If errors are found, the first error of the worst severity is returned. For example, if a warning (A) and two errors (B and C) are found, the first error (B) is returned as the image exit status. The image exit status is placed in the DCL symbol $STATUS at image exit.

The ANALYZE/IMAGE command provides the following information:

- Image type—Identifies whether the image is executable or shareable.

- Image transfer addresses—Identify the addresses to which control is passed at image execution time.

- Image version—Identifies the revision level of the image.

- Patch information—Indicates whether the image has been patched (changed without having been recompiled or reassembled and relinked). If a patch is present, the actual patch code can be displayed.

- Location of the debugger symbol table (DST)—Identifies the location of the DST in the image file. DST information is present only in executable images that have been linked with the /DEBUG or the /TRACEBACK command qualifier.

- Location of the global symbol table (GST)—Identifies the location of the GST in the image file. GST information is present only in shareable image files.

- Image section descriptors (ISD)—Identify portions of the image binary contents that are grouped in clusters according to their attributes. An ISD contains information that the image activator needs when it initializes the address space for an image. For example, an ISD tells whether the ISD is shareable, whether it is readable or writable, whether it is based or position independent, and how much memory should be allocated.

- Fixup vectors—Contain information that the image activator needs to ensure the position independence of shareable image references.

- System version categories—For an image that is linked against the system symbol table, displays both the values of the system version categories for which the image was linked originally and the values for the system that is currently running. You can use these values to identify changes in the system since the image was linked last.

The ANALYZE/IMAGE command has command qualifiers and positional qualifiers. By default, if you do not specify any positional qualifiers (for example, /GST or /HEADER), the entire image is analyzed. If you do specify a positional qualifier, the analysis excludes all other positional qualifiers except the /HEADER qualifier (which is always enabled) and any qualifier that you request explicitly.

QUALIFIERS

/FIXUP_SECTION
Positional qualifier.

Specifies that the analysis should include all information in the fixup section of the image.

If you specify the /FIXUP_SECTION qualifier after the ANALYZE/IMAGE command, the fixup section of each image file in the parameter list is analyzed.

If you specify the /FIXUP_SECTION qualifier after a file specification, only the information in the fixup section of that image file is analyzed.

/GST
Positional qualifier.

Specifies that the analysis should include all global symbol table records. This qualifier is valid only for shareable images.

If you specify the /GST qualifier after the ANALYZE/IMAGE command, the global symbol table records of each image file in the parameter list are analyzed.

If you specify the /GST qualifier after a file specification, only the global symbol table records of that file are analyzed.

/HEADER
Positional qualifier.

Specifies that the analysis should include all header items and image section descriptions. The image header items are analyzed always.

/INTERACTIVE
/NOINTERACTIVE (default)

Specifies whether the analysis is interactive. In interactive mode, as each item is analyzed, the results are displayed on the screen and you are asked whether you want to continue.

/OUTPUT=filespec

Identifies the output file for storing the results of the image analysis. No wildcard characters are allowed in the file specification. If you specify a file type and omit the file name, the default file name ANALYZE is used. The default file type is ANL. If you omit the qualifier, the results are output to the current SYS$OUTPUT device.

/PATCH_TEXT
Positional qualifier.

Specifies that the analysis include all patch text records. If you specify the /PATCH_TEXT qualifier after the ANALYZE/IMAGE command, the patch text records of each image file in the parameter list are analyzed.

If you specify the /PATCH_TEXT qualifier after a file specification, only the patch text records of that file are analyzed.

EXAMPLES

1 $ ANALYZE/IMAGE LINEDT

The ANALYZE/IMAGE command in this example produces a description and an error analysis of the image LINEDT.EXE. Output is sent to the current SYS$OUTPUT device. By default, the entire image is analyzed.

2 $ ANALYZE/IMAGE/OUTPUT=LIALPHEX/FIXUP_SECTION/PATCH_TEXT LINEDT, ALPHA

The ANALYZE/IMAGE command in this example produces a description and an error analysis of the fixup sections and patch text records of LINEDT.EXE and ALPHA.EXE in file LIALPHEX.ANL. Output is sent to the file LIALPHEX.ANL.

ANALYZE/MEDIA

Invokes the Bad Block Locator Utility, which analyzes block-addressable devices and records the location of blocks that cannot reliably store data. For a complete description of the Bad Block Locator Utility, see the *VMS Bad Block Locator Utility Manual.*

FORMAT   **ANALYZE/MEDIA** *device*

ANALYZE/OBJECT

Analyzes the contents of an object file and checks for any obvious errors. The /OBJECT qualifier is required. (Use the ANALYZE/IMAGE command to analyze the contents of an image file.)

FORMAT	**ANALYZE/OBJECT** *filespec[,...]*

PARAMETER	***filespec[,...]*** Specifies the object files or object module libraries you want analyzed (the default file type is OBJ). Use commas (,) or plus signs (+) to separate file specifications. Wildcard characters (* and %) are allowed in the file specification.

DESCRIPTION	The ANALYZE/OBJECT command describes the contents of one or more object modules contained in one or more files. It also performs a partial error analysis. This analysis determines whether the records in an object module conform in content, format, and sequence to the specifications of the VMS Object Language.

ANALYZE/OBJECT is intended primarily for programmers of compilers, debuggers, or other software involving VMS object modules. It checks that the object language records generated by the object modules are acceptable to the VMS Linker, and it identifies certain errors in the file. It also provides a description of the records in the object file or object module library. For more information on the VMS linker and on the VMS Object Language, refer to the *VMS Linker Utility Manual.*

The ANALYZE/OBJECT command analyzes the object modules in order, record by record, from the first to the last record in the object module. Fields in each record are analyzed in order from the first to the last field in the record. After the object module is analyzed, you should compare the content and format of each type of record to the required content and format of that record as described by the VMS Object Language. This comparison is particularly important if the analysis output contains a diagnostic message.

Linking an object module differs from analyzing an object module. Object language commands are not executed in an analysis, but they are executed in a linking operation. As a result, even if the analysis is error free, the linking operation may not be. In particular, the analysis does not check the following:

- That data arguments in TIR commands are in the correct format.

- That "Store Data" TIR commands are storing within legal address limits.

Therefore, as a final check, you should still link an object module whose analysis is error free.

If an error is found, however, the first error of the worst severity that is discovered is returned. For example, if a warning (A) and two errors (B and C) are signaled, then the first error (B) is returned as the image exit status, which is placed in the DCL symbol $STATUS at image exit.

ANALYZE/OBJECT uses positional qualifiers; that is, qualifiers whose function depends on their position in the command line. When a positional qualifier precedes all of the input files in a command line, it affects all input files. For example, the following command line requests that the analysis include the global symbol directory records in files A, B, and C:

```
$ ANALYZE/OBJECT/GSD A,B,C
```

Conversely, when a positional qualifier is associated with only one file in the parameter list, only that file is affected. For example, the following command line requests that the analysis include the global symbol directory records in file B only:

```
$ ANALYZE/OBJECT A,B/GSD,C
```

Typically, all records in an object module are analyzed. However, when the /DBG, /EOM, /GSD, /LNK, /MHD, /TBT, or /TIR qualifier is specified, only the record types indicated by the qualifiers are analyzed. All other record types are ignored.

By default, the analysis includes all record types unless you explicitly request a limited analysis using appropriate qualifiers.

Note: **End-of-module (EOM) records and module header (MHD) records are always analyzed, no matter which qualifiers you specify.**

QUALIFIERS **/DBG**
Positional qualifier.

Specifies that the analysis should include all debugger information records. If you want the analysis to include debugger information for all files in the parameter list, insert the /DBG qualifier immediately following the /OBJECT qualifier. If you want the analysis to include debugger information selectively, insert the /DBG qualifier immediately following each of the selected file specifications.

/EOM
Positional qualifier.

Specifies that the analysis should be limited to MHD records, EOM records, and records explicitly specified by the command. If you want this to apply to all files in the parameter list, insert the /EOM qualifier immediately following the /OBJECT qualifier.

To make the /EOM qualifier applicable selectively, insert it immediately following each of the selected file specifications.

Note: **End-of-module records can be EOM or EOMW records. See the VMS Linker~Utility Manual for more information.**

/GSD
Positional qualifier.

Specifies that the analysis should include all global symbol directory (GSD) records.

If you want the analysis to include GSD records for each file in the parameter list, specify the /GSD qualifier immediately following the /OBJECT qualifier.

If you want the analysis to include GSD records selectively, insert the /GSD qualifier immediately following each of the selected file specifications.

/INCLUDE[=(module[,...])]
When the specified file is an object module library, use this qualifier to list selected object modules within the library for analysis. If you omit the list or specify an asterisk (*), all modules are analyzed. If you specify only one module, you can omit the parentheses.

/INTERACTIVE
/NOINTERACTIVE (default)
Controls whether the analysis occurs interactively. In interactive mode, as each record is analyzed, the results are displayed on the screen, and you are asked whether you want to continue.

/LNK
Positional qualifier.

Specifies that the analysis should include all link option specification (LNK) records.

If you want the analysis to include LNK records for each file in the parameter list, specify the /LNK qualifier immediately following the /OBJECT qualifier.

If you want the analysis to include LNK records selectively, insert the /LNK qualifier immediately following each of the selected file specifications.

/MHD
Positional qualifier.

Specifies that the analysis should be limited to MHD records, EOM records, and records explicitly specified by the command. If you want this analysis to apply to all files in the parameter list, insert the /MHD qualifier immediately following the /OBJECT qualifier.

To make the /MHD qualifier applicable selectively, insert immediately following each of the selected file specifications.

/OUTPUT[=filespec]
Directs the output of the object analysis (the default is SYS$OUTPUT). If you specify a file type and omit the file name, the default file name ANALYZE is used. The default file type is ANL.

No wildcard characters are allowed in the file specification.

/TBT
Positional qualifier.

Specifies that the analysis should include all module traceback (TBT) records.

If you want the analysis to include TBT records for each file in the parameter list, specify the /TBT qualifier immediately following the /OBJECT qualifier.

If you want the analysis to include TBT records selectively, insert the /TBT qualifier immediately following each of the selected file specifications.

/TIR
Positional qualifier.

Specifies that the analysis should include all text information and relocation (TIR) records.

If you want the analysis to include TIR records for each file in the parameter list, specify the /TIR qualifier immediately following the /OBJECT qualifier.

If you want the analysis to include TIR records selectively, insert the /TIR qualifier immediately following the selected file specifications.

EXAMPLES

1   `$ ANALYZE/OBJECT/INTERACTIVE  LINEDT`

In this example, the ANALYZE/OBJECT command produces a description and a partial error analysis of the object file LINEDT.OBJ. By default, all types of records are analyzed. Output is to the terminal, because the /INTERACTIVE qualifier has been used. As each item is analyzed, the utility displays the results on the screen and asks if you want to continue.

2   `$ ANALYZE/OBJECT/OUTPUT=LIOBJ/DBG  LINEDT`

In this example, the ANALYZE/OBJECT command analyzes only the debugger information records of the file LINEDT.OBJ. Output is to the file LIOBJ.ANL.

ANALYZE/PROCESS_DUMP

Invokes the VMS Debugger to analyze a process dump file that was created when an image failed during execution. (Use the /DUMP qualifier with the RUN or the SET PROCESS command to generate a dump file.) For a complete description of the debugger (including information about the DEBUG command), see the *VMS Debugger Manual.*

Requires read (R) access to the dump file.

FORMAT

ANALYZE/PROCESS_DUMP *dump-file*

PARAMETER

dump-file
Specifies the dump file to be analyzed with the debugger.

DESCRIPTION

The ANALYZE/PROCESS_DUMP command examines the dump file of an image that failed during execution. The VMS Debugger is invoked automatically. To cause a dump file to be created for a process, you must use the /DUMP qualifier with the RUN command when invoking the image, or you must use the SET PROCESS/DUMP command before invoking the image.

QUALIFIERS

/FULL
Displays all known information about the failing process.

/IMAGE=image-name
/NOIMAGE
Specifies the image whose symbols are to be used in analyzing the dump. If you use the /NOIMAGE qualifier, no symbols are taken from any image. By default, symbols are taken from the image with the same name as the image that was running at the time of the dump.

/INTERACTIVE
/NOINTERACTIVE (default)
Causes the display of information to pause when your terminal screen is filled. Press the Return key to display additional information. By default, the display is continuous.

/MISCELLANEOUS
Displays all the miscellaneous information in the dump.

/OUTPUT=filespec
Writes the information to the specified file. By default, the information is written to the current SYS$OUTPUT device. No wildcard characters are allowed in the file specification.

/RELOCATION

Displays the addresses to which data structures saved in the dump are
mapped in P0 space. (Examples of such data structures are the stacks.)
The data structures in the dump must be mapped into P0 space so that
the debugger can use those data structures in P1 space.

EXAMPLE

```
$ ANALYZE/PROCESS/FULL ZIPLIST

R0 = 00018292   R1 = 8013DE20   R2 = 7FFE6A40   R3 = 7FFE6A98
R4 = 8013DE20   R5 = 00000000   R6 = 7FFE7B9A   R7 = 0000F000
R8 = 00000000   R9 = 00000000   R10 = 00000000  R11 = 00000000
SP = 7FFAEF44   AP = 7FFAEF48   FP  = 7FFAEF84
FREE_P0_VA  00001600      FREE_P1_VA  7FFAC600
Active ASTs  00            Enabled ASTs 0F
Current Privileges  FFFFFF80  1010C100
Event Flags  00000000  E0000000
Buffered I/O count/limit  6/6
Direct I/O count/limit    6/6
File count/limit          27/30
Process count/limit       0/0
Timer queue count/limit   10/10
AST count/limit           6/6
Enqueue count/limit       30/30
Buffered I/O total 7      Direct I/O total 18
Link Date  27-DEC-1990 15:02:00.48   Patch Date  17-NOV-1990 00:01:53.71
ECO Level  0030008C  00540040  00000000  34303230
Kernel stack 00000000 pages at 00000000 moved to 00000000
Exec stack 00000000 pages at 00000000 moved to 00000000
Vector page 00000001 page at 7FFEFE00 moved to 00001600
PIO (RMS) area 00000005 pages at 7FFE1200 moved to 00001800
Image activator context 00000001 page at 7FFE3400 moved to 00002200
User writable context 0000000A pages at 7FFE1C00 moved to 00002400
Creating a subprocess
        VAX DEBUG Version 5.4
DBG>
```

This example shows the output of the ANALYZE/PROCESS command
when used with the /FULL qualifier. The file specified, ZIPLIST, contains
the dump of a process that encountered a fatal error. The DBG> prompt
indicates that the debugger is ready to accept commands.

ANALYZE/RMS_FILE

Invokes the Analyze/RMS_File Utility, which is used to inspect and analyze
the internal structure of a VMS RMS file. The /RMS_FILE qualifier is required.
For a complete description of the Analyze/RMS_File Utility, see the *VMS
Analyze/RMS_File Utility Manual.*

FORMAT   **ANALYZE/RMS_FILE**   *filespec[,...]*

ANALYZE/SYSTEM

Invokes the System Dump Analyzer Utility, which analyzes a running system. The /SYSTEM qualifier is required. For a complete description of the System Dump Analyzer Utility, see the *VMS System Dump Analyzer Utility Manual.*

FORMAT   **ANALYZE/SYSTEM**

APPEND

Adds the contents of one or more specified input files to the end of the specified output file.

FORMAT **APPEND** *input-filespec[,...] output-filespec*

PARAMETERS *input-filespec[,...]*
Specifies the names of one or more input files to be appended. Multiple input files are appended to the output file in the order specified. If you specify more than one input file, separate the file specifications with either commas (,) or plus signs (+).

Wildcard characters (* and %) are allowed in the input file specifications.

output-filespec
Specifies the name of the file to which the input files will be appended.

You must specify at least one field in the output file specification. If you do not specify a device or directory, the APPEND command uses the current default device and directory. Other unspecified fields default to the corresponding fields of the first input file specification.

If you use the asterisk (*) wildcard character in any fields of the output file specification, the APPEND command uses the corresponding field of the input file specification. If you are appending more than one input file, the APPEND command uses the corresponding fields from the first input file.

DESCRIPTION The APPEND command is similar in syntax and function to the COPY command. Normally, the APPEND command adds the contents of one or more files to the end of an existing file without incrementing the version number. The /NEW_VERSION qualifier causes the APPEND command to create a new output file if no file with that name exists.

Note that there are special considerations for using the APPEND command with DECwindows compound documents. For more information, see the *Guide to VMS File Applications*.

QUALIFIERS */ALLOCATION=number-of-blocks*
Forces the initial allocation of the output file to the specified number of 512-byte blocks. If you do not specify the /ALLOCATION qualifier, or if you specify it without the *number-of-blocks* parameter, the initial allocation of the output file is determined by the size of the input file.

The allocation size is applied only if a new file is actually created by using the /NEW_VERSION qualifier.

/BACKUP

Modifies the time value specified with the /BEFORE or the /SINCE qualifier. The /BACKUP qualifier selects files according to the dates of their most recent backups. This qualifier is incompatible with the /CREATED, /EXPIRED, and /MODIFIED qualifiers, which also allow you to select files according to time attributes. If you specify none of these four time qualifiers, the default is the /CREATED qualifier.

/BEFORE[=time]

Selects only those files dated prior to the specified time. You can specify time as an absolute time, as a combination of absolute and delta times, or as one of the following keywords: TODAY (default), TOMORROW, or YESTERDAY. Specify one of the following qualifiers with the /BEFORE qualifier to indicate the time attribute to be used as the basis for selection: /BACKUP, /CREATED (default), /EXPIRED, or /MODIFIED.

For complete information on specifying time values, see the *VMS DCL Concepts Manual*.

/BY_OWNER[=uic]

Selects only those files whose owner user identification code (UIC) matches the specified owner UIC. The default UIC is that of the current process.

Specify the UIC by using standard UIC format as described in the *VMS DCL Concepts Manual*.

/CONFIRM
/NOCONFIRM (default)

Controls whether a request is issued before each append operation to confirm that the operation should be performed on that file. The following responses are valid:

YES	NO	QUIT
TRUE	FALSE	Ctrl/Z
1	0	ALL
	Return	

You can use any combination of uppercase and lowercase letters for word responses. Word responses can be abbreviated to one or more letters (for example, T, TR, or TRU for TRUE), but these abbreviations must be unique. Affirmative answers are YES, TRUE, and 1. Negative answers include: NO, FALSE, 0, and pressing the Return key. Entering QUIT or pressing Ctrl/Z indicates that you want to stop processing the command at that point. When you respond by entering ALL, the command continues to process, but no further prompts are given. If you type a response other than one of those in the list, DCL issues an error message and redisplays the prompt.

/CONTIGUOUS
/NOCONTIGUOUS

Specifies that the output file must occupy physically contiguous disk blocks. By default, the APPEND command creates an output file in the same format as the corresponding input file and does not report an error if not enough space exists for a contiguous allocation. This qualifier is relevant only with the /NEW_VERSION qualifier.

If an input file is contiguous, the APPEND command attempts to create a contiguous output file, but does not report an error if there is not enough space. If you append multiple input files of different formats, the output file may or may not be contiguous. Use the /CONTIGUOUS qualifier to ensure that the output file is contiguous.

/CREATED (default)

Modifies the time value specified with the /BEFORE or the /SINCE qualifier. The /CREATED qualifier selects files based on their dates of creation. This qualifier is incompatible with the /BACKUP, /EXPIRED, and /MODIFIED qualifiers, which also allow you to select files according to time attributes. If you specify none of these four time qualifiers, the default is the /CREATED qualifier.

/EXCLUDE=(filespec[,...])

Excludes the specified files from the append operation. You can include a directory but not a device in the file specification. Wildcard characters (* and %) are allowed in the file specification. However, you cannot use relative version numbers to exclude a specific version. If you specify only one file, you can omit the parentheses.

/EXPIRED

Modifies the time value specified with the /BEFORE or the /SINCE qualifier. The /EXPIRED qualifier selects files according to their expiration dates. (The expiration date is set with the SET FILE/EXPIRATION_DATE command.) The /EXPIRED qualifier is incompatible with the /BACKUP, /CREATED, and /MODIFIED qualifiers, which also allow you to select files according to time attributes. If you specify none of these four time qualifiers, the default is the /CREATED qualifier.

/EXTENSION=number-of-blocks

Specifies the number of blocks to be added to the output file each time the file is extended. When you specify the /EXTENSION qualifier, the /NEW_VERSION qualifier is assumed and need not be typed on the command line. This qualifier is relevant only with the /NEW_VERSION qualifier.

The extension value is applied only if a new file is actually created.

/LOG
/NOLOG (default)

Controls whether the APPEND command displays the file specifications of each file appended. If the /LOG qualifier is specified, the command displays the file specifications of the input and output files as well as the number of blocks or records appended after each append operation.

/MODIFIED

Modifies the time value specified with the /BEFORE or the /SINCE qualifier. The /MODIFIED qualifier selects files according to the dates on which they were last modified. This qualifier is incompatible with the /BACKUP, /CREATED, and /EXPIRED qualifiers, which also allow you to select files according to time attributes. If you specify none of these four time modifiers, the default is the /CREATED qualifier.

/NEW_VERSION
/NONEW_VERSION (default)

Controls whether the APPEND command creates a new output file if the specified output file does not exist. (By default, the specified output file already exists.) If the specified output file does not already exist, use the /NEW_VERSION qualifier to create a new output file. If the output file does exist, the /NEW_VERSION qualifier is ignored and the input file is appended to the output file.

/PROTECTION=(ownership[:access][,...])

Specifies protection for the output file. Specify *ownership* as system (S), owner (O), group (G), or world (W) and *access* as read (R), write (W), execute (E), or delete (D). The default protection, including any protection attributes not specified, is that of the existing output file. If no output file exists, the current default protection applies. This qualifier is relevant only with the /NEW_VERSION qualifier.

For more information on specifying protection codes, see the *VMS DCL Concepts Manual*.

/READ_CHECK
/NOREAD_CHECK (default)

Reads each record in the input files twice to verify that it has been read correctly.

/SINCE[=time]

Selects only those files dated after the specified time. You can specify time as an absolute time, as a combination of absolute and delta times, or as one of the following keywords: TODAY (default), TOMORROW, or YESTERDAY. Specify one of the following qualifiers with the /SINCE qualifier to indicate the time attribute to be used as the basis for selection: /BACKUP, /CREATED (default), /EXPIRED, or /MODIFIED.

For complete information on specifying time values, see the *VMS DCL Concepts Manual*.

/WRITE_CHECK
/NOWRITE_CHECK (default)

Reads each record in the output file after the record is written to verify that it was appended successfully and that the output file can subsequently be read without error.

EXAMPLES

1 $ APPEND TEST3.DAT TESTALL.DAT

The APPEND command appends the contents of the file TEST3.DAT from the default disk and directory to the file TESTALL.DAT, also located on the default disk and directory.

2
```
$ APPEND/NEW_VERSION/LOG *.TXT    MEM.SUM
%APPEND-I-CREATED, USE$:[MAL]MEM.SUM;1 created
%APPEND-S-COPIED, USE$:[MAL]A.TXT;2 copied to USE$:[MAL]MEM.SUM;1 (1 block)
%APPEND-S-APPENDED, USE$:[MAL]B.TXT;3 appended to USE$:[MAL]MEM.SUM;1 (3 records)
%APPEND-S-APPENDED, USE$:[MAL]G.TXT;7 appended to USE$:[MAL]MEM.SUM;1 (51 records)
```

The APPEND command appends all files with file types of TXT to a file named MEM.SUM. The /LOG qualifier requests a display of the specifications of each input file appended. If the file MEM.SUM does not exist, the APPEND command creates it, as the output shows. The number of blocks or records shown in the output refers to the source file and not to the target file total.

3
```
$ APPEND/LOG A.DAT, B.MEM    C.*
%APPEND-S-APPENDED, USE$:[MAL]A.DAT;4 appended to USE$:[MAL]C.DAT;4 (2 records)
%APPEND-S-APPENDED, USE$:[MAL]B.MEM;5 appended to USE$:[MAL]C.DAT;4 (29 records)
```

The APPEND command appends the files A.DAT and B.MEM to the file C.DAT, which must already exist.

4
```
$ APPEND/LOG A.*    B.*
%APPEND-S-APPENDED, USE$:[MAL]A.DAT;5 appended to USE$:[MAL]B.DAT;1 (5 records)
%APPEND-S-APPENDED, USE$:[MAL]A.DOC;2 appended to USE$:[MAL]B.DAT;1 (1 record)
```

Both the input and output file specifications contain wildcard characters in the file type field. The APPEND command appends each file with a file name of A to an existing file with B as its file name. The file type of the first input file located determines the output file type.

5
```
$ APPEND BOSTON"JOHN_SMITH YANKEE"::DEMO01.DAT, DEMO2.DAT
$ _To:    DALLAS::DISK1:[MODEL.TEST]TEST.DAT
```

This APPEND command adds the contents of the files DEMO1.DAT and DEMO2.DAT at remote node BOSTON to the end of the file TEST.DAT at remote node DALLAS.

ASSIGN

Creates a logical name and assigns an equivalence string, or a list of strings, to the specified logical name. If you specify an existing logical name, the new equivalence name replaces the existing equivalence name.

FORMAT   **ASSIGN**   *equivalence-name[,...] logical-name[:]*

PARAMETERS   *equivalence-name[,...]*
Specifies a character string of 1 to 255 characters. Defines the equivalence name, usually a file specification, device name, or other logical name, to be associated with the logical name in the specified logical name table. If the string contains other than uppercase alphanumeric, dollar sign ($), or underscore (_) characters, enclose it in quotation marks (" "). Use two consecutive quotation marks ("") to denote an actual quotation mark within the string. Specifying more than one equivalence name for a logical name creates a search list.

When you specify an equivalence name that will be used as a file specification, you must include the punctuation marks (colons [:], brackets [[ ]], and periods [.]) that would be required if the equivalence name were used directly as a file specification. Therefore, if you specify a device name as an equivalence name, terminate the device name with a colon.

The ASSIGN command allows you to assign the same logical name to more than one equivalence name. When you specify more than one equivalence name for a logical name, you create a search list. For more information on search lists, see the *VMS DCL Concepts Manual*.

logical-name[:]
Specifies the logical name string, which is a character string containing up to 255 characters. You choose a logical name to represent the equivalence name in the specified logical name table.

If the string contains other than uppercase alphanumeric, dollar sign, or underscore characters, enclose it in quotation marks. Use two consecutive quotation marks to denote an actual quotation mark. If you terminate the logical-name parameter with a colon, the system removes the colon before placing the name in a logical name table. (This differs from the DEFINE command, which saves the colon.) If the logical name is to be entered into the process directory (LNM$PROCESS_DIRECTORY) or system directory (LNM$SYSTEM_DIRECTORY) logical name tables, then the name may only have from 1 to 31 alphanumeric characters (including the dollar sign and underscore). By default, the logical name is placed in the process logical name table.

If the logical name contains any characters other than alphanumeric characters, the dollar sign, or the underscore, enclose the name in quotation marks. If the logical name contains quotation marks, enclose the name in quotation marks and use two consecutive quotation marks in the places where you want one set of quotation marks to occur. Note that if you enclose a name in quotation marks, the case of alphabetic characters is preserved.

ASSIGN

DESCRIPTION
The ASSIGN command creates an entry in a logical name table by defining a logical name to stand for one or more equivalence names. An equivalence name can be a device name, another logical name, a file specification, or any other string.

To specify the logical name table where you want to enter a logical name, use the /PROCESS, /JOB, /GROUP, /SYSTEM, or /TABLE qualifier. If you enter more than one of these qualifiers, only the last one entered is accepted. If you do not specify a table, the default is /TABLE=LNM$PROCESS (or /PROCESS).

To specify the access mode of the logical name you are creating, use the /USER_MODE, the /SUPERVISOR_MODE, or the /EXECUTIVE_MODE qualifier. If you enter more than one of these qualifiers, only the last one entered is accepted. If you do not specify an access mode, then a supervisor-mode name is created. You can create a logical name in the same mode as the table in which you are placing the name or in an outer mode. (User mode is the outermost mode; executive mode is the innermost mode.)

You can enter more than one logical name with the same name in the same logical name table, as long as each name has a different access mode. (However, if an existing logical name within a table has the NO_ALIAS attribute, you cannot use the same name to create a logical name in an outer mode in this table.)

If you create a logical name with the same name, in the same table, and in the same mode as an existing name, the new logical name assignment replaces the existing assignment.

You can also use the DEFINE command to create logical names. To delete a logical name from a table, use the DEASSIGN command.

Note: **Avoid assigning a logical name that matches the file name of an executable image in SYS$SYSTEM:. Such an assignment will prohibit you from invoking that image.**

For additional information on how to create and use logical names, see the *VMS DCL Concepts Manual*.

QUALIFIERS
/EXECUTIVE_MODE
Requires SYSNAM (system logical name) privilege.

Creates an executive-mode logical name. If you specify executive mode, but do not have SYSNAM privilege, a supervisor-mode logical name is created. The mode of the logical name must be the same as or external to (less privileged than) the mode of the table in which you are placing the name.

/GROUP

Requires SYSPRV (system privilege) or GRPNAM (group logical name) privilege.

Places the logical name in the group logical name table. Other users who have the same group number in their user identification codes (UICs) can access the logical name. The /GROUP qualifier is synonymous with the /TABLE=LNM$GROUP qualifier.

/JOB

Places the logical name in the jobwide logical name table. All processes within the same job tree as the process creating the logical name can access the logical name. The /JOB qualifier is synonymous with the /TABLE=LNM$JOB qualifier.

/LOG (default)
/NOLOG

Displays a message when a new logical name supersedes an existing name.

/NAME_ATTRIBUTES[=(keyword[,...])]

Specifies the attributes for a logical name. By default, no attributes are set. You can specify the following keywords for attributes:

CONFINE   Does not copy the logical name into a spawned subprocess; this keyword is relevant only for logical names in a private table.

NO_ALIAS   Prohibits creation of logical names with the same name in an outer (less privileged) access mode within the specified table. If another logical name with the same name and an outer access mode already exists in this table, the name is deleted.

If you specify only one keyword, you can omit the parentheses. Only the attributes you specify are set.

/PROCESS (default)

Places the logical name in the process logical name table. The /PROCESS qualifier is synonymous with the /TABLE=LNM$PROCESS qualifier.

/SUPERVISOR_MODE (default)

Creates a supervisor-mode logical name in the specified table.

/SYSTEM

Requires SYSNAM (system logical name) or SYSPRV (system privilege) privilege.

Places the logical name in the system logical name table. All system users can access the logical name. The /SYSTEM qualifier is synonymous with the /TABLE=LNM$SYSTEM qualifier.

/TABLE=name

Requires write (W) access to the table if the table is shareable.

Specifies the logical name table in which the logical name is to be entered. You can use the /TABLE qualifier to specify a user-defined logical name table (created with the CREATE/NAME_TABLE command); to specify the process, job, group, or system logical name tables; or to specify the process or system logical name directory tables.

If you specify the table name using a logical name that has more than one translation, the logical name is placed in the first table found. For example, if you specify ASSIGN/TABLE=LNM$FILE_DEV and LNM$FILE_DEV is equated to LNM$PROCESS, LNM$JOB, LNM$GROUP, and LNM$SYSTEM, then the logical name is placed in LNM$PROCESS.

If you do not explicitly specify the /TABLE qualifier, the default is the /TABLE=LNM$PROCESS qualifier.

/TRANSLATION_ATTRIBUTES[=(keyword[,...])]

Equivalence-name qualifier.

Specifies attributes of the equivalence-name parameter. Possible keywords are as follows:

CONCEALED   Indicates that the equivalence string is the name of a concealed device.

When a concealed device name is defined, the system displays the logical name, rather than the equivalence string, in messages that refer to the device. If you specified the CONCEALED attribute, then the equivalence string must be a physical device name.

TERMINAL   Indicates that the equivalence string should not be translated iteratively; logical name translation should terminate with the current equivalence string.

If you specify only one keyword, you can omit the parentheses. Only the attributes you specify are set.

Note that different equivalence strings of the same logical name can have different translation attributes specified.

/USER_MODE

Creates a user-mode logical name in the specified table.

If you specify a user-mode logical name in the process logical name table, that logical name is used for the execution of a single image only; user-mode entries are deleted from the logical name table when any image executing in the process exits; that is, after any DCL command that executes an image or user program completes execution.

EXAMPLES

1   `$ ASSIGN $DISK1:[ACCOUNTS.MEMOS] MEMOSD`

> The ASSIGN command in this example equates the partial file specification $DISK1:[ACCOUNTS.MEMOS] to the logical name MEMOSD.

2   `$ ASSIGN/USER_MODE $DISK1:[ACCOUNTS.MEMOS]WATER.TXT TM1`

> The ASSIGN command in this example equates the logical name TM1 to a file specification. After the next image runs, the logical name is deassigned automatically.

3   `$ ASSIGN XXX1:[CHARLES]  CHARLIE`
      `$ PRINT  CHARLIE:TEST.DAT`
      `Job 274 entered on queue SYS$PRINT`

> The ASSIGN command in this example associates the logical name CHARLIE with the directory name [CHARLES] on the disk XXX1. Subsequent references to the logical name CHARLIE result in the correspondence between the logical name CHARLIE and the disk and directory specified. The PRINT command queues a copy of the file XXX1:[CHARLES]TEST.DAT to the system printer.

4   `$ ASSIGN YYY2:  TEMP:`
      `$ SHOW LOGICAL TEMP`
        `"TEMP" = "YYY2:" (LNM$PROCESS_TABLE)`
      `$ DEASSIGN TEMP`

> The ASSIGN command in this example equates the logical name TEMP to the device YYY2. TEMP is created in supervisor mode and placed in the process logical name table. The SHOW LOGICAL command verifies that the logical name assignment was made. Note that the logical name TEMP was terminated with a colon in the ASSIGN command, but that the command interpreter deleted the colon before placing the name in the logical name table. Thus, you can specify TEMP without a colon in the subsequent DEASSIGN command. You should omit the colon in the SHOW LOGICAL command (for example, SHOW LOGICAL TEMP).

5   `$ MOUNT TTT1: MASTER TAPE`
      `$ ASSIGN TAPE:NAMES.DAT PAYROLL`
      `$ RUN PAYROLL`
        `.`
        `.`
        `.`

> The MOUNT command in this example establishes the logical name TAPE for the device TTT1, which has the volume labeled MASTER mounted on it. The ASSIGN command equates the logical name PAYROLL with the file named NAMES.DAT on the logical device TAPE. Thus, an OPEN request in a program referring to the logical name PAYROLL results in the correspondence between the logical name PAYROLL and the file NAMES.DAT on the tape whose volume label is MASTER.

```
6  $  CREATE/NAME_TABLE TABLE1
   $  ASSIGN/TABLE=LNM$PROCESS_DIRECTORY TABLE1,-
  _$  LNM$PROCESS,LNM$JOB,LNM$GROUP,LNM$SYSTEM LNM$FILE_DEV
   $  ASSIGN/TABLE=TABLE1 -
  _$  /TRANSLATION_ATTRIBUTES=CONCEALED  DBA1:  WORK_DISK
```

The CREATE/NAME_TABLE command in this example creates the process private logical name table TABLE1.

The first ASSIGN command ensures that TABLE1 is searched first in any logical name translation of a file specification or device name (because TABLE1 is the first item in the equivalence string for the logical name LNM$FILE_DEV, which determines the default search sequence of logical name tables whenever a device or file specification is translated).

The second ASSIGN command assigns the logical name WORK_DISK to the physical device DBA1, and places the name in TABLE1. The logical name has the concealed attribute. Therefore, the logical name WORK_DISK will be displayed in system messages.

```
7  $  ASSIGN/TABLE=LNM$PROCESS/TABLE=LNM$GROUP  DBA0:  SYSFILES
   $  SHOW LOGICAL  SYSFILES
     "SYSFILES" = "DBA0:" (LNM$GROUP_000240)
```

The ASSIGN command in this example contains conflicting qualifiers. When you specify conflicting qualifiers, the ASSIGN command uses the last qualifier specified. The response from the SHOW LOGICAL command indicates that the name was placed in the group logical name table.

```
8  $  ASSIGN/TABLE=LNM$GROUP 'F$TRNLNM("SYS$COMMAND")' TERMINAL
   %DCL-I-SUPERSEDE, previous value of TERMINAL has been superseded
```

The ASSIGN command in this example uses the lexical function F$TRNLNM to translate the logical name SYS$COMMAND and use the result as the equivalence name for the logical name TERMINAL. The message from the ASSIGN command indicates that an entry for the logical name TERMINAL already existed in the group logical name table, and that the new entry has replaced the previous one.

If this command is used in a LOGIN.COM file, the entry for TERMINAL will be redefined at the beginning of each terminal session. The current process and any subprocesses it creates can execute images that use the logical name TERMINAL to write messages to the current terminal device.

```
9  $  ASSIGN DALLAS::DMA1:  DATA
```

The ASSIGN command in this example associates the logical name DATA with the device specification DMA1 on remote node DALLAS. Subsequent references to the logical name DATA result in references to the disk on the remote node.

```
🔟  $  CREATE AVERAGE.COM
    $  ASSIGN/USER_MODE SYS$COMMAND:   SYS$INPUT
    $  EDIT/EDT AVERAGE.FOR
    $  FORTRAN AVERAGE
    $  LINK AVERAGE
    $  RUN AVERAGE
    87
    80
    90
    9999
    $  EXIT
    Ctrl/Z
    $  @AVERAGE.COM
```

The CREATE command in this example creates the command procedure AVERAGE.COM. Then the command procedure is executed.

The command procedure uses the ASSIGN command with the /USER_ MODE qualifier to change temporarily the value of SYS$INPUT. When the EDT editor is invoked, it accepts input from the terminal. This allows you to create or modify the program AVERAGE.FOR interactively.

When you exit from EDT, SYS$INPUT is reassigned to its original value (the input stream provided by the command procedure). Thus, when the program AVERAGE.FOR is ready to accept input, it looks for that input in the command procedure.

ASSIGN/MERGE

Removes all jobs from one queue and merges them into another existing queue. This command does not affect jobs that are executing.

Requires OPER (operator) privilege or execute (E) access to both queues.

FORMAT **ASSIGN/MERGE** *target-queue[:] source-queue[:]*

PARAMETERS *target-queue[:]*
Specifies the name of the queue into which the jobs are being merged.

source-queue[:]
Specifies the name of the queue from which the jobs are being removed.

DESCRIPTION The ASSIGN/MERGE command removes the pending jobs in one queue and places them in another queue. This command does not affect any executing jobs in either the target queue or the source queue. Jobs currently running in the source queue complete in that queue. This command is generally used with printer queues, although it can be used with batch queues.

The ASSIGN/MERGE command is particularly useful when a line printer malfunctions. By entering the ASSIGN/MERGE command, you can reroute existing jobs to a different printing device. To perform the merge operation without losing or disrupting any jobs, stop the source queue with the STOP/QUEUE/NEXT command. Then enter the STOP/QUEUE /REQUEUE command to ensure that the current job on the source queue is requeued for processing on the target queue. (If the STOP/QUEUE /REQUEUE command fails to requeue the job, use the STOP/QUEUE /RESET command to regain control of the queue.) Once you enter the STOP commands, enter the ASSIGN/MERGE command.

EXAMPLE

```
$  STOP/QUEUE/NEXT LPB0
$  STOP/QUEUE/REQUEUE=LPA0 LPB0
$  ASSIGN/MERGE LPA0 LPB0
```

In this example, the STOP/QUEUE/NEXT command prevents another job from executing on queue LPB0. The STOP/QUEUE/REQUEUE command requeues the current job running on LPB0 to the target queue LPA0. The ASSIGN/MERGE command removes the remaining jobs from the LPB0 printer queue and places them in the LPA0 printer queue.

ASSIGN/QUEUE

Assigns, or redirects, a logical queue to a single execution queue. The ASSIGN/QUEUE command can be used only with printer or terminal queues.

Requires OPER (operator) privilege or execute (E) access to both queues.

FORMAT   **ASSIGN/QUEUE**   *queue-name[:]*
                            logical-queue-name[:]

PARAMETERS   *queue-name[:]*
Specifies the name of the execution queue. The queue cannot be a logical queue, a generic queue, or a batch queue.

logical-queue-name[:]
Specifies the name of the logical queue.

DESCRIPTION   The ASSIGN/QUEUE command sets up a one-to-one correspondence between a logical queue and an execution queue. Jobs submitted to the logical queue are always queued to the specified execution queue for eventual printing.

When you enter the ASSIGN/QUEUE command, the logical queue cannot be running.

Once you initialize a logical queue, use the ASSIGN/QUEUE command to associate the logical queue with an existing execution queue. You must perform the following tasks to set up a logical queue:

1   Initialize the logical queue with the INITIALIZE/QUEUE command. (Do not use the /START qualifier.)

2   Assign the logical queue name to an existing execution queue.

3   Start the logical queue with the START/QUEUE command.

After you enter the START/QUEUE command for the logical queue, jobs can be sent to the logical queue for processing.

EXAMPLES

1 ```
$ INITIALIZE/QUEUE/DEFAULT=FLAG=ONE/START LPA0
$ INITIALIZE/QUEUE TEST_QUEUE
$ ASSIGN/QUEUE LPA0 TEST_QUEUE
$ START/QUEUE TEST_QUEUE
```

This example first initializes and starts the printer queue LPA0. The LPA0 queue is set to have a flag page precede each job. The second INITIALIZE/QUEUE command creates the logical queue TEST_QUEUE.

The ASSIGN/QUEUE command assigns the logical queue TEST_QUEUE to the printer queue LPA0. The START/QUEUE command starts the logical queue.

**2**    `$  INITIALIZE/QUEUE/START LPB0`

The ASSIGN/QUEUE command is not needed in this example because a logical queue is not being initialized. A printer queue is being initialized; LPB0 is the name of a line printer. After you enter the INITIALIZE /QUEUE/START command, jobs can be queued to LPB0 for printing.

# ATTACH

Transfers control from your current process (which then hibernates) to the specified process.

**The ATTACH and SPAWN commands cannot be used if your terminal has an associated mailbox.**

---

**FORMAT**     **ATTACH** *[process-name]*

---

**PARAMETER**     *process-name*
Specifies the name of a parent process or spawned subprocess to which control passes. The process must already exist, be part of your current job, and share the same input stream as your current process. However, the process cannot be your current process or a subprocess created with the /NOWAIT qualifier.

Process names can contain from 1 to 15 alphanumeric characters. If a connection to the specified process cannot be made, an error message is displayed.

The *process-name* parameter is incompatible with the /IDENTIFICATION qualifier.

---

**DESCRIPTION**     The ATTACH command allows you to connect your input stream to another process. You can use the ATTACH command to change control from one subprocess to another subprocess or to the parent process.

When you enter the ATTACH command, the parent or "source" process is put into hibernation, and your input stream is connected to the specified destination process. You can use the ATTACH command to connect to a subprocess that is part of a current job left hibernating as a result of the SPAWN/WAIT command or another ATTACH command as long as the connection is valid. (No connection can be made to the current process, to a process that is not part of the current job, or to a process that does not exist. If any of these connections are attempted, an error message is displayed.)

You can also use the ATTACH command in conjunction with the SPAWN /WAIT command to return to a parent process without terminating the created subprocess. See the description of the SPAWN command for more details.

---

**QUALIFIER**     */IDENTIFICATION=pid*
Specifies the process identification (PID) of the process to which terminal control will be transferred. Leading zeros can be omitted. The /IDENTIFICATION qualifier is incompatible with the *process-name* parameter.

If you omit the /IDENTIFICATION qualifier, you must specify a process name.

## EXAMPLES

**1**    `$ ATTACH JONES_2`

The ATTACH command transfers the terminal's control to the subprocess JONES_2.

**2**    `$ ATTACH/IDENTIFICATION=30019`

The ATTACH command switches control from the current process to a process having the PID 30019. Notice that because the /IDENTIFICATION qualifier is specified, the *process-name* parameter is omitted.

# BACKUP

Invokes the Backup Utility (BACKUP) to perform one of the following BACKUP operations:

- Make copies of disk files.

- Save disk files as data in a file created by BACKUP on disk or magnetic tape. (Files created by BACKUP are called save sets.)

- Restore disk files from a BACKUP save set.

- Compare disk files or files in a BACKUP save set with other disk files.

- List information about files in a BACKUP save set to an output device or file.

Note that standalone BACKUP cannot be invoked this way, but must be bootstrapped in order to run. For a complete description of BACKUP and information on standalone BACKUP, see the *VMS Backup Utility Manual.*

**FORMAT**     **BACKUP**  *input-specifier output-specifier*

# CALL

Transfers control to a labeled subroutine within a command procedure.

---

**FORMAT**     **CALL**   *label [parameter[,...]]*

---

**PARAMETERS**   *label*
Specifies a label of 1 to 255 alphanumeric characters that appears as the
first item on a command line. A label cannot contain embedded blanks.
When the CALL command is executed, control passes to the command
following the specified label.

The label can precede or follow the CALL statement in the current
command procedure. A label in a command procedure must be terminated
with a colon ( : ). Labels for subroutines must be unique.

Labels declared in inner procedure levels are inaccessible from outer
levels, as in the following example:

```
$CALL B
$SUBROUTINE A
$  B: SUBROUTINE
$   ENDSUBROUTINE
$ENDSUBROUTINE
```

In this example, the label B in subroutine A is inaccessible from the outer
procedure level.

parameter[,...]
Specifies from one to eight optional parameters to pass to the command
procedure. Use two consecutive quotation marks ("") to specify a null
parameter. The parameters assign character string values to the symbols
named P1, P2, and so on in the order of entry, to a maximum of eight.
The symbols are local to the specified command procedure. Separate each
parameter with one or more blanks.

You can specify a parameter with a character string value containing
alphanumeric or special characters, with the following restrictions:

- The command interpreter converts alphabetic characters to uppercase
  and uses blanks to delimit each parameter. To pass a parameter that
  contains embedded blanks or lowercase letters, enclose the parameter
  in quotation marks (" ").

- If the first parameter begins with a slash (/), you must enclose the
  parameter in quotation marks.

- To pass a parameter that contains quotation marks and spaces, enclose
  the entire string in quotation marks and use two consecutive quotation
  marks within the string. For example:

  ```
  $ CALL SUB1 "Never say ""quit"""
  ```

When control transfers to SUB1, the parameter P1 is equated to the following string:

```
Never say "quit"
```

If a string contains quotation marks and does not contain spaces, the quotation marks are preserved in the string and the letters within the quotation marks remain in lowercase. For example:

```
$ CALL SUB2 abc"def"ghi
```

When control transfers to SUB2, the parameter P1 is equated to the string:

ABCdefGHI

To use a symbol as a parameter, enclose the symbol in single quotation marks (' ') to force symbol substitution. For example:

```
$ NAME = "JOHNSON"
$ CALL INFO 'NAME'
```

The single quotation marks cause the value "JOHNSON" to be substituted for the symbol "NAME". Therefore, the parameter "JOHNSON" is passed as P1 to the subroutine INFO.

DESCRIPTION

The CALL command transfers control to a labeled subroutine within a command procedure. The CALL command is similar to the @ (execute procedure) command in that it creates a new procedure level. The advantage of the CALL command is that it does not require files to be opened and closed to process the procedure. Using the CALL command also makes managing a set of procedures easier because they can all exist in one file rather than in several files.

When you use the CALL command to transfer control to a subroutine, a new procedure level is created and the symbols P1 to P8 are assigned the values of the supplied arguments. Execution then proceeds until an EXIT command is encountered. At this point, control is transferred to the command line following the CALL command.

Procedures can be nested to a maximum of 32 levels, which includes any combination of command procedure and subroutine calls. Local symbols and labels defined within a nested subroutine structure are treated the same way as if the routines had been invoked with the @ command; that is, labels are valid only for the subroutine level in which they are defined.

Local symbols defined in an outer subroutine level are available to any subroutine levels at an inner nesting level; that is, the local symbols can be read, but they cannot be written to. If you assign a value to a symbol that is local to an outer subroutine level, a new symbol is created at the current subroutine level. However, the symbol in the outer procedure level is not modified.

The SUBROUTINE and ENDSUBROUTINE commands define the beginning and end of a subroutine. The label defining the entry point to the subroutine must appear either immediately before the SUBROUTINE command or on the same command line.

A subroutine can have only one entry point. The subroutine must begin with the SUBROUTINE command as the first executable statement. If an EXIT command is not specified in the procedure, the ENDSUBROUTINE command functions as an EXIT command.

The SUBROUTINE command performs two different functions depending on the context in which it is executed. If executed as the result of a CALL command, it initiates a new procedure level, defines the parameters P1 to P8 as specified in the CALL statement, and begins execution of the subroutine. If the SUBROUTINE verb is encountered in the execution flow of the procedure without having been invoked by a CALL command, all the commands following the SUBROUTINE command are skipped until the corresponding ENDSUBROUTINE command is encountered.

Note: **The SUBROUTINE and ENDSUBROUTINE commands cannot be abbreviated to fewer than 4 characters.**

QUALIFIER

/OUTPUT=filespec
Writes all output to the file or device specified. By default, the output is written to the current SYS$OUTPUT device and the output file type is LIS. System responses and error messages are written to SYS$COMMAND as well as to the specified file. If you specify /OUTPUT, the qualifier must immediately follow the CALL command. No wildcard characters are allowed in the output file specification.

You can also redefine SYS$OUTPUT to redirect the output from a command procedure. If you place the following command as the first line in a command procedure, output will be directed to the file you specify:

$ DEFINE SYS$OUTPUT filespec

When the procedure exits, SYS$OUTPUT is restored to its original equivalence string. This produces the same result as using the /OUTPUT qualifier when you execute the command procedure.

EXAMPLE

```
$
$! CALL.COM
$
$! Define subroutine SUB1
$!
$ SUB1: SUBROUTINE
     .
     .
     .
$ CALL SUB2 !Invoke SUB2 from within SUB1
     .
     .
     .
$ @FILE   !Invoke another procedure command file
     .
     .
     .
$ EXIT
$ ENDSUBROUTINE !End of SUB1 definition
$!
$! Define subroutine SUB2
$!
$ SUB2: SUBROUTINE
     .
     .
     .
$ EXIT
$ ENDSUBROUTINE !End of SUB2 definition
$!
$! Start of main routine. At this point, both SUB1 and SUB2
$! have been defined but none of the previous commands have
$! been executed.
$!
$ START:
$ CALL/OUTPUT=NAMES.LOG SUB1 "THIS IS P1"
     .
     .
     .
$ CALL SUB2 "THIS IS P1" "THIS IS P2"
     .
     .
     .
$ EXIT   !Exit this command procedure file
```

The command procedure in this example shows how to use the CALL command to transfer control to labeled subroutines. The example also shows that you can call a subroutine or another command file from within a subroutine. The CALL command invokes the subroutine SUB1, directing output to the file NAMES.LOG and allowing other users write (W) access to the file. The subroutine SUB2 is called from within SUB1. The procedure executes SUB2 and then uses the @ (execute procedure) command to invoke the command procedure FILE.COM. When all the commands in SUB1 have executed, the CALL command in the main procedure calls SUB2 a second time. The procedure continues until SUB2 has executed.

CANCEL

Cancels wakeup requests for a specified process, including wakeup requests scheduled with either the RUN command or the $SCHDWK system service.

Requires one of the following:

- **Ownership of the process.**

- **GROUP privilege to cancel scheduled wakeup requests for processes in the same group but not owned by you.**

- **WORLD privilege to cancel scheduled wakeup requests for any process in the system.**

FORMAT

CANCEL *[[node-name::]process-name]*

PARAMETERS

node-name::

The name of the node on which the specified process is running. The node name can have as many as six alphanumeric characters. The two colons (::) count for two additional characters, for a total of eight.

You cannot specify a node name on a different VAXcluster from the current process.

process-name

The name of the process for which wakeup requests are to be canceled. The process name can have up to 15 alphanumeric characters.

The specified process must be in the same group as the current process.

DESCRIPTION

The CANCEL command cancels scheduled wakeup requests for the specified process.

The CANCEL command does not delete the specified process. If the process is executing an image when the CANCEL command is issued for it, the process hibernates instead of exiting after the image completes execution.

To delete a hibernating process for which wakeup requests have been canceled, use the STOP command. You can determine whether a subprocess has been deleted by entering the SHOW PROCESS command with the /SUBPROCESSES qualifier.

A local process name can look like a remote process name. Therefore, if you specify ATHENS::SMITH, the system checks for a process named ATHENS::SMITH on the local node before checking node ATHENS for a process named SMITH.

You also can use the /IDENTIFICATION=pid qualifier to specify a process name. If you use the /IDENTIFICATION qualifier and the *process-name* parameter together, the qualifier overrides the parameter. If you do not specify either the *process-name* parameter or the /IDENTIFICATION qualifier, the CANCEL command cancels scheduled wakeup requests for the current (that is, the issuing) process.

QUALIFIER

/IDENTIFICATION=pid

Identifies the process by its process identification (PID). You can omit leading zeros when you specify the PID.

EXAMPLES

1   `$ CANCEL CALENDAR`

The CANCEL command in this example cancels a wakeup request for a process named CALENDAR (which continues to hibernate until it is deleted with the STOP command).

2
```
$ RUN/SCHEDULE=14:00 STATUS
%RUN-S-PROC_ID, identification of created process is 0013012A
    .
    .
    .
$ CANCEL/IDENTIFICATION=13012A
```

The RUN command in this example creates a process to execute the image STATUS. The process hibernates and is scheduled to be awakened at 14:00. Before the process is awakened, the CANCEL command cancels the wakeup request.

3
```
$ RUN/PROCESS_NAME=LIBRA/INTERVAL=1:00    LIBRA
%RUN-S-PROC_ID, identification of created process is 00130027
    .
    .
    .
$ CANCEL LIBRA
$ STOP LIBRA
```

The RUN command in this example creates a subprocess named LIBRA to execute the image LIBRA.EXE at hourly intervals.

Subsequently, the CANCEL command cancels the wakeup request. The process continues to exist, but in a state of hibernation, until the STOP command deletes it.

CLOSE

Closes a file opened with the OPEN command and deassigns the associated logical name.

FORMAT   **CLOSE**   *logical-name[:]*

PARAMETER   *logical-name[:]*
Specifies the logical name assigned to the file when it was opened with the OPEN command.

DESCRIPTION   Files that are opened for reading or writing at the command level remain open until closed with the CLOSE command, or until the process terminates. If a command procedure that opens a file terminates without closing the open file, the file remains open; the command interpreter does not automatically close it.

QUALIFIERS   */ERROR=label*
Specifies a label in the command procedure to receive control if the close operation results in an error. Overrides any ON condition action specified. If an error occurs and the target label is successfully given control, the global symbol $STATUS retains the code for the error that caused the error path to be taken.

/LOG (default)
/NOLOG
Generates a warning message when you attempt to close a file that was not opened by DCL. If you specify the /ERROR qualifier, the /LOG qualifier has no effect. If the file has not been opened by DCL, the error branch is taken and no message is displayed.

EXAMPLES

1
```
$ OPEN/READ INPUT_FILE   TEST.DAT
$ READ_LOOP:
$ READ/END_OF_FILE=NO_MORE   INPUT_FILE   DATA_LINE
   .
   .
   .
$ GOTO READ_LOOP
$ NO_MORE:
$ CLOSE INPUT_FILE
```

> The OPEN command in this example opens the file TEST.DAT and assigns it the logical name of INPUT_FILE. The /END_OF_FILE qualifier on the READ command requests that, when the end-of-file (EOF) is reached, the command interpreter should transfer control to the line at the label NO_MORE. The CLOSE command closes the input file.

2
```
$  @READFILE
 Ctrl/Y
$  STOP
$  SHOW LOGICAL/PROCESS
   .
   .
   .
  "INFILE" = " _DB1"
  "OUTFILE" = " _DB1"
$  CLOSE INFILE
$  CLOSE OUTFILE
```

> In this example, pressing Ctrl/Y interrupts the execution of the command procedure READFILE.COM. Then, the STOP command stops the procedure. The SHOW LOGICAL/PROCESS command displays the names that currently exist in the process logical name table. Among the names listed are the logical names INFILE and OUTFILE, assigned by OPEN commands in the procedure READFILE.COM.

> The CLOSE commands close these files and deassign the logical names.

CONNECT

Connects your physical terminal to a virtual terminal that is connected to another process.

You must connect to a virtual terminal that is connected to a process with your user identification code (UIC). No other physical terminals may be connected to the virtual terminal.

FORMAT

CONNECT *virtual-terminal-name*

PARAMETER

virtual-terminal-name

Specifies the name of the virtual terminal to which you are connecting. A virtual terminal name always begins with the letters VTA. To determine the name of the virtual terminal that is connected to a process, enter the SHOW USERS command.

DESCRIPTION

The CONNECT command connects you to a separate process, as opposed to the SPAWN and ATTACH commands, which create and attach subprocesses.

The CONNECT command is useful when you are logged in to the system using telecommunications lines. If there is noise over the line and you lose the carrier signal, your process does not terminate. After you log in again, you can reconnect to the original process and log out of your second process.

To use the CONNECT command, the virtual terminal feature must be enabled for your system with the System Generation Utility (SYSGEN). If virtual terminals are allowed on your system, then the SET TERMINAL /DISCONNECT/PERMANENT command is used to enable the virtual terminal characteristic for a particular physical terminal. When this characteristic is enabled, a virtual terminal will be created when a user logs in to the physical terminal. The physical terminal is connected to the virtual terminal, which is in turn connected to the process.

When the connection between the physical terminal and the virtual terminal is broken, you are logged out of your current process (and any images that the process is executing stop running) unless you have specified the /NOLOGOUT qualifier.

If you have specified the /NOLOGOUT qualifier, the process remains connected to the virtual terminal. If the process is executing an image, it continues until the process needs terminal input or attempts to write to the terminal. At that point, the process waits until the physical terminal is reconnected to the virtual terminal.

You can connect to a virtual terminal even if you are not currently using a virtual terminal. However, to log out of your current process you must use the CONNECT command with the /LOGOUT qualifier. If you connect to a virtual terminal from another virtual terminal, you can save your current process by using the /NOLOGOUT qualifier.

QUALIFIERS	***/CONTINUE***
	/NOCONTINUE (default)

Controls whether the CONTINUE command is executed in the current process just before connecting to another process. This qualifier allows an interrupted image to continue processing after you connect to another process.

The /CONTINUE qualifier is incompatible with the /LOGOUT qualifier.

/LOGOUT (default)
/NOLOGOUT

Logs out your current process when you connect to another process using a virtual terminal.

When you enter the CONNECT command from a process that is not connected to a virtual terminal, you must specify the /LOGOUT qualifier. Otherwise, DCL displays an error message.

The /LOGOUT qualifier is incompatible with the /CONTINUE qualifier.

EXAMPLES

1
```
$  RUN AVERAGE
   Ctrl/Y
$  CONNECT/CONTINUE VTA72
```

In this example, you use the RUN command to execute the image AVERAGE.EXE. You enter this command from a terminal that is connected to a virtual terminal. Next, you press Ctrl/Y to interrupt the image. After you interrupt the image, enter the CONNECT command with the /CONTINUE qualifier. This operation issues the CONTINUE command, so the image continues to run and connects you to another virtual terminal. You can reconnect to the process later.

```
2   $ SHOW USERS
            VMS Interactive Users
            19-APR-1990 15:25:30.75
       Total number of interactive users = 5
    Username       Process Name      PID      Terminal
    REICH          Steve          2040055C  VTA267:        TXC13:
    GLASS          Phil           20400560  VTA270:        LTA102:
    ADAMS          ADAMS          20400551  VTA261:        TTC7:
    DUFAY          DUFAY          2040056D  VTA272:        Disconnected
    DUFAY          _VTA273:       2040056E  VTA273:        TTB5:
    $ CONNECT VTA273
      DUFAY          logged out at 19-APR-1990 15:26:56.53
    $
```

This example shows how to reconnect to your original process after you have lost the carrier signal. First, you must log in again and create a new process. After you log in, enter the SHOW USERS command to determine the virtual terminal name for your initial process. Then enter the CONNECT command to connect to the virtual terminal associated with your original process. The process from which you enter the CONNECT command is logged out because you have not specified any qualifiers.

When you reconnect to the original process, you continue running the image that you were running when you lost the carrier signal. In this example, the user DUFAY was at interactive level when the connection was broken.

CONTINUE

Resumes execution of a DCL command, a program, or a command procedure that was interrupted by pressing Ctrl/Y or Ctrl/C. You cannot resume execution of the image if you have entered a command that executes another image or if you have invoked a command procedure.

FORMAT   **CONTINUE**

PARAMETERS   *None.*

DESCRIPTION   The CONTINUE command enables you to resume processing an image or a command procedure that was interrupted by pressing Ctrl/Y or Ctrl/C. You cannot resume execution of the image if you have entered a command that executes another image or if you have invoked a command procedure. However, you can use CONTINUE after commands that do not execute separate images; for a list of these commands, see the *VMS DCL Concepts Manual.*

You can abbreviate the CONTINUE command to a single letter, C.

The CONTINUE command serves as the target command of an IF or ON command in a command procedure. The CONTINUE command is also a target command when it follows a label that is the target of a GOTO command. In addition, you can use the CONTINUE command to resume processing of a program that has executed either a VAX FORTRAN PAUSE statement or a VAX COBOL-74 STOP literal statement.

EXAMPLES

1   ```
$ RUN MYPROGRAM_A
Ctrl/Y
$ SHOW TIME
   19-APR-1990 13:40:12
$ CONTINUE
```

In this example, the RUN command executes the program MYPROGRAM_A. While the program is running, pressing Ctrl/Y interrupts the image. The SHOW TIME command requests a display of the current date and time. The CONTINUE command resumes the image.

CONTINUE

2 $ ON SEVERE_ERROR THEN CONTINUE

> In this example, the command procedure statement requests the command
> interpreter to continue executing the procedure if any warning, error, or
> severe error status value is returned from the execution of a command or
> program. This ON statement overrides the default action, which is to exit
> from a procedure following errors or severe errors.

CONVERT

Invokes the Convert Utility, which copies records from one file to another and changes the organization and format of the input file to those of the output file. For a complete description of the Convert Utility, see the *VMS Convert and Convert/Reclaim Utility Manual.*

FORMAT   **CONVERT** *input-filespec[,...] output-filespec*

CONVERT/DOCUMENT

Converts a revisable format file to another revisable or final form file.

You can use this command only if you have VMS DECwindows installed on your system.

FORMAT   **CONVERT/DOCUMENT** *input-filespec output-filespec*

DESCRIPTION   The CONVERT/DOCUMENT command invokes the CDA Converter to convert a revisable format file to another revisable or final form file.

PARAMETERS   *input-filespec*
Specifies the file to be converted. The default file type is DDIF.

output-filespec
Specifies the name of the converted file. The default file type is DDIF.

QUALIFIERS   */FORMAT=format-name*
Specifies the encoding format of the input or output file. The default format is DDIF for both input and output.

Input formats bundled with the VMS operating system and their default file extensions are as follows:

Input Format	File Extension
DDIF	.DDIF
DTIF	.DTIF
TEXT	.TXT

Output formats bundled with the VMS operating system and their default file extensions are as follows:

Output Format	File Extension
DDIF	.DDIF
DTIF	.DTIF
TEXT	.TXT
PS	.PS
ANALYSIS	.CDA$ANALYSIS

Digital's CDA Converter Library is a layered product that provides additional input and output formats. Independent software vendors who write DDIF- and DTIF-conforming applications and front and back ends also provide input and output formats that are layered on the VMS

operating system. Contact your system manager for a complete list of input and output formats available on your system.

/OPTIONS=options-filename

Specifies a file that contains processing options for both input and output. The default file extension for a VMS options file is .CDA$OPTIONS.

Creating the Options File

You can create an options file that contains all the input and output processing options to be applied during the conversion of the input file to the output file. These processing options affect how your input file is processed and how your output file is created or displayed.

Each line of the options file specifies a format keyword (for example, PS for PostScript) that can be followed optionally by _INPUT or _OUTPUT to restrict the option to the front or back end. The second keyword is a valid processing option preceded by one or more spaces, tabs, or a slash (/) and can contain upper- and lowercase alphabetic characters (alphabetic case is not significant), digits (0–9), dollar signs ($), and underscores (_). If an option requires you to specify a value, the option keyword can be separated from the value by one or more spaces or tabs, or by an equal sign (=). Each line can be preceded optionally by spaces and tabs.

The following example is a typical entry in an options file:

```
PS PAPER_HEIGHT 10
```

In this example, the extension _OUTPUT is not required for the format keyword, since PostScript is available only as an output format. The value specified for PAPER_HEIGHT is in inches by default.

To specify several options for the same input or output format, you must specify each option on a separate line. The CDA Converter checks the input format and the output format you specified on the command line and, if the processing options in your options file are valid for the input or output format, the options are applied during the conversion of your file. If you specify an invalid option for an input or output format or an invalid value for an option, the CDA Converter returns an error message. Each input and output format that supports processing options specifies any restrictions or special formats required when specifying processing options.

Processing options available for several of the file formats that are bundled with VMS are listed in the following sections.

Text Back-End Processing Options

The text back-end converter supports the following options:

- *ASCII_FALLBACK [ON,OFF]*

  Causes the text back-end converter to output text in 7-bit ASCII. The fallback representation of the characters is described in the ASCII standard. If this option is not specified, the default is OFF; if this option is specified without a value, the default is ON.

- *CONTENT_MESSAGES [ON,OFF]*

  Causes the text back-end converter to put a message in the output file each time a nontext element is encountered in the in-memory CDA structures. If this option is not specified, the default is OFF; if this option is specified without a value, the default is ON.

CONVERT/DOCUMENT

- *HEIGHT value*

  Specifies the maximum number of lines per page in your text output file. If you specify zero, the number of lines per page will correspond to the height specified in your document. If you additionally specify OVERRIDE_FORMAT, or if the document has no inherent page size, the document is formatted to the height value specified by this option. The default height is 66 lines.

- *OVERRIDE_FORMAT [ON,OFF]*

  Causes the text back-end converter to ignore the document formatting information included in your document, so that the text is formatted in a single large galley per page that corresponds to the size of the page as specified by the HEIGHT and WIDTH processing options. If this option is not specified, the default is OFF; if this option is specified without a value, the default is ON.

- *SOFT_DIRECTIVES [ON,OFF]*

  Causes the text back-end converter to obey the soft directives contained in the document when creating your text output file. If this option is not specified, the default is OFF; if this option is specified without a value, the default is ON.

- *WIDTH value*

  Specifies the maximum number of columns of characters per page in your text output file. If you specify zero, the number of columns per page will correspond to the width specified in your document. If you additionally specify OVERRIDE_FORMAT, or if the document has no inherent page size, the document is formatted to the value specified by this processing option. If any lines of text exceed this width value, the additional columns are truncated. The default width is 80 characters.

PostScript Back-End Processing Options

The PostScript back-end converter supports the following processing options:

- *PAPER_SIZE size*

  Specifies the size of the paper to be used when formatting the resulting PostScript output file. Valid values for the *size* argument are as follows:

Keyword	Size
A0	841 x 1189 millimeters (33.13 x 46.85 inches)
A1	594 x 841 millimeters (23.40 x 33.13 inches)
A2	420 x 594 millimeters (16.55 x 23.40 inches)
A3	297 x 420 millimeters (11.70 x 16.55 inches)
A4	210 x 297 millimeters (8.27 x 11.70 inches)
A	8.5 x 11 inches
B	11 x 17 inches
C	17 x 22 inches

Keyword	Size
D	22 x 34 inches
E	34 x 44 inches
LEDGER	11 x 17 inches
LEGAL	8.5 x 14 inches
LETTER	8.5 x 11 inches
LP	13.7 x 11 inches
VT	8 x 5 inches

The A paper size (8.5 x 11 inches) is the default.

- *PAPER_HEIGHT height*

  Specifies a paper size other than one of the predefined values provided. The default paper height is 11 inches.

- *PAPER_WIDTH width*

  Specifies a paper size other than one of the predefined sizes provided. The default paper width is 8.5 inches.

- *PAPER_TOP_MARGIN top-margin*

  Specifies the width of the margin provided at the top of the page. The default value is 0.25 inch.

- *PAPER_BOTTOM_MARGIN bottom-margin*

  Specifies the width of the margin provided at the bottom of the page. The default value is 0.25 inch.

- *PAPER_LEFT_MARGIN left-margin*

  Specifies the width of the margin provided on the left-hand side of the page. The default value is 0.25 inch.

- *PAPER_RIGHT_MARGIN right-margin*

  Specifies the width of the margin provided on the right-hand side of the page. The default value is 0.25 inch.

- *PAPER_ORIENTATION orientation*

  Specifies the paper orientation to be used in the output PostScript file. The valid values for the *orientation* argument are as follows:

Keyword	Meaning
PORTRAIT	The page is oriented so that the larger dimension is parallel to the vertical axis.
LANDSCAPE	The page is oriented so that the larger dimension is parallel to the horizontal axis.

The default is PORTRAIT.

- *EIGHT_BIT_OUTPUT [ON,OFF]*

  Specifies whether the PostScript back-end converter should use 8-bit output. The default value is ON.

- *LAYOUT [ON,OFF]*

  Specifies whether the PostScript back-end converter processes the layout specified in the DDIF document. The default value is ON.

- *OUTPUT_BUFFER_SIZE output-buffer-size*

  Specifies the size of the output buffer. The value you specify must be within the range 64 to 256. The default value is 132.

- *PAGE_WRAP [ON,OFF]*

  Specifies whether the PostScript back-end converter performs page wrapping of any text that would exceed the bottom margin. The default value is ON.

- *SOFT_DIRECTIVES [ON,OFF]*

  Specifies whether the PostScript back-end converter processes soft directives in the DDIF file in order to format output. (Soft directives specify such formatting commands as new line, new page, and tab.) If the PostScript back-end converter processes soft directives, the output file will look more like you intended. The default value is ON.

- *WORD_WRAP [ON,OFF]*

  Specifies whether the PostScript back-end converter performs word wrapping of any text that would exceed the right margin. The default value is ON. If you specify OFF, the PostScript back-end converter allows text to exceed the right margin.

Analysis Back-End Processing Option

The analysis back-end converter produces an analysis of the CDA in-memory structure in the form of text output showing the named objects and values stored in the document. This is useful for debugging DDIF application programs.

The analysis back-end converter supports an INHERITANCE processing option, which specifies that the analysis is shown with attribute inheritance enabled. Inherited attributes are marked by "[default]" in the output.

Domain Conversion Processing Options

When you are converting any table format to any document format, you can specify the following processing options using a format name of DTIF_TO_DDIF:

- *COLUMN_TITLE*

  Displays the column titles as contained in the column attributes centered at the top of the column.

- *CURRENT_DATE*

  Displays the current date and time in the bottom left corner of the page. The value is formatted according to the document's specification for a default date and time.

- *DOCUMENT_DATE*

  Displays the document date and time as contained in the document header in the top left corner of the page. The value is formatted according to the document's specification for a default date and time.

- *DOCUMENT_TITLE*

  Displays the document title or titles as contained in the document header centered at the top of the page, one string per line.

- *PAGE_NUMBER*

  Displays the current page number in the top right corner of the page.

- *PAPER_SIZE size*

  Specifies the size of the paper to be used when formatting the resulting PostScript output file. The values are the same as those for the PostScript back-end converter.

- *PAPER_HEIGHT height*

  Specifies a paper size other than one of the predefined values provided. The default paper height is 11 inches.

- *PAPER_WIDTH width*

  Specifies a paper size other than one of the predefined sizes provided. The default paper width is 8.5 inches.

- *PAPER_TOP_MARGIN top-margin*

  Specifies the width of the margin provided at the top of the page. The default value is 0.25 inch.

- *PAPER_BOTTOM_MARGIN bottom-margin*

  Specifies the width of the margin provided at the bottom of the page. The default value is 0.25 inch.

- *PAPER_LEFT_MARGIN left-margin*

  Specifies the width of the margin provided on the left-hand side of the page. The default value is 0.25 inch.

- *PAPER_RIGHT_MARGIN right-margin*

  Specifies the width of the margin provided on the right-hand side of the page. The default value is 0.25 inch.

- *PAPER_ORIENTATION orientation*

  Specifies the paper orientation to be used in the output file. The values are the same as those for the PostScript back-end converter.

EXAMPLE

```
$ CONVERT/DOCUMENT -

_$ /OPTIONS=OPTIONS.CDA$OPTIONS -

_$ FOOBAR.DTIF/FORMAT=DTIF -
_$ MOOMAR.DDIF/FORMAT=DDIF
```

This command converts an input file named FOOBAR.DTIF, which has the DTIF format, to an output file named MOOMAR.DDIF, which has the DDIF format. The specified options file is named OPTIONS.CDA$OPTIONS.

CONVERT/RECLAIM

Invokes the Convert/Reclaim Utility, which makes empty buckets in Prolog 3 indexed files available so that new records can be written in them. The /RECLAIM qualifier is required. For a complete description of the Convert /Reclaim Utility, see the *VMS Convert and Convert/Reclaim Utility Manual*.

FORMAT   **CONVERT/RECLAIM** *filespec*

COPY

Creates a new file from one or more existing files. The COPY command can do the following:

- Copy an input file to an output file.
- Concatenate two or more input files into a single output file.
- Copy a group of input files to a group of output files.

FORMAT   **COPY**   *input-filespec[,...] output-filespec*

PARAMETERS   *input-filespec[,...]*
Specifies the name of an existing file to be copied. Wildcard characters (* and %) are allowed. If you do not specify the device or directory, the COPY command uses your current default device and directory. If you specify more than one file, separate the file specifications with either commas (,) or plus signs (+).

output-filespec
Specifies the name of the output file into which the input is copied.

You must specify at least one field in the output file specification. If you do not specify the device or directory, the COPY command uses your current default device and directory. The COPY command replaces any other missing fields (file name, file type, version number) with the corresponding field of the input file specification. If you specify more than one input file, the COPY command generally uses the fields from the first input file to determine any missing fields in the output file.

You can use the asterisk (*) wildcard character in place of any two of the following: the file name, the file type, or the version number. The COPY command uses the corresponding field in the related input file to name the output file.

DESCRIPTION   The COPY command creates a new file from one or more existing files. If you do not specify the device or directory, the COPY command uses your current default device and directory. The COPY command can do the following:

- Copy an input file to an output file.
- Concatenate two or more input files into a single output file.
- Copy a group of input files to a group of output files.

The COPY command, by default, creates a single output file. When you specify more than one input file, the first input file is copied to the output file, and each subsequent input file is appended to the end of the output file. If a field of the output file specification is missing or contains an asterisk wildcard character, the COPY command uses the corresponding field from the first, or only, input file to name the output file.

If you specify multiple input files with maximum record lengths, the COPY command gives the output file the maximum record length of the first input file. If the COPY command encounters a record in a subsequent input file that is longer than the maximum record length of the output file, it issues a message noting the incompatible file attributes and begins copying the next file.

To create multiple output files, specify multiple input files and use at least one of the following:

- An asterisk wildcard character in the output directory specification, file name, file type, or version number field

- Only a node name, a device name, or a directory specification as the output file specification

- The /NOCONCATENATE qualifier

When the COPY command creates multiple output files, it uses the corresponding field from each input file in the output file name. You also can use the wildcard character in the output file specification to have COPY create more than one output file. For example:

```
$ COPY A.A;1, B.B;1 *.C
```

This COPY command creates the files A.C;1 and B.C;1 in the current default directory. When you specify multiple input and output files you can use the /LOG qualifier to verify that the files were copied as you intended.

Note that there are special considerations for using the COPY command with DECwindows compound documents. For more information, see the *Guide to VMS File Applications*.

Version Numbers

If you do not specify version numbers for input and output files, the COPY command (by default) assigns a version number to the output files that is either of the following:

- The version number of the input file

- A version number one greater than the highest version number of an existing file with the same file name and file type

When you specify the output file version number by an asterisk wildcard character, the COPY command uses the version numbers of the associated input files as the version numbers of the output files.

If you specify the output file version number by an explicit version number, the COPY command uses that number for the output file specification. If a higher version of the output file exists, the COPY command issues a warning message and copies the file. If an equal version of the output file exists, the COPY command issues a message and does *not* copy the input file.

File Protection and Creation/Revision Dates

The COPY command considers an output file to be new when you specify any portion of the output file name explicitly. The COPY command sets the creation date for a new file to the current time and date.

If you specify the output file by one or more wildcard characters, the COPY command uses the creation date of the input file.

The COPY command always sets the revision date of the output file to the current time and date; it sets the backup date to zero. The file system assigns the output file a new expiration date. (The file system sets expiration dates if retention is enabled; otherwise, it sets expiration dates to zero.)

The protection and access control list (ACL) of the output file is determined by the following parameters, in the following order:

- Protection of previously existing versions of the output file

- Default Protection and ACL of the output directory

- Process default file protection

(Note that the BACKUP command takes the creation and revision dates as well as the file protection from the input file.)

Use the /PROTECTION qualifier to change the output file protection.

Normally, the owner of the output file will be the same as the creator of the output file. However, if a user with extended privileges creates the output file, the owner will be the owner of the parent directory or of a previous version of the output file if one exists.

Extended privileges include any of the following:

- SYSPRV (system privilege) or BYPASS

- System user identification code (UIC)

- GRPPRV (group privilege) if the owner of the parent directory (or previous version of the output file) is in the same group as the creator of the new output file

- An identifier (with the resource attribute) representing the owner of the parent directory (or the previous version of the output file)

Copying Directory Files

If you copy a file that is a directory, the COPY command creates a new *empty* subdirectory of the named directory. The COPY command does *not* copy any files from the named directory to the new subdirectory. For example:

```
$ COPY [SMITH]CATS.DIR [JONES]
```

This COPY command creates the new empty subdirectory [JONES]CATS.DIR. Once the COPY command creates the new subdirectory [JONES]CATS.DIR, you can copy the files in the directory [SMITH]CATS.DIR.

QUALIFIERS	**/ALLOCATION=number-of-blocks**

Forces the initial allocation of the output file to the specified number of 512-byte blocks. If you do not specify the /ALLOCATION qualifier, or if you specify it without the *number-of-blocks* parameter, the initial allocation of the output file is determined by the size of the input file being copied.

/BACKUP

Modifies the time value specified with the /BEFORE or the /SINCE qualifier. The /BACKUP qualifier selects files according to the dates of their most recent backups. This qualifier is incompatible with the /CREATED, /EXPIRED, and /MODIFIED qualifiers, which also allow you to select files according to time attributes. If you specify none of these four time qualifiers, the default is the /CREATED qualifier.

/BEFORE[=time]

Selects only those files dated prior to the specified time. You can specify time as an absolute time, as a combination of absolute and delta times, or as one of the following keywords: TODAY (default), TOMORROW, or YESTERDAY. Specify one of the following qualifiers with the /BEFORE qualifier to indicate the time attribute to be used as the basis for selection: /BACKUP, /CREATED (default), /EXPIRED, or /MODIFIED.

For complete information on specifying time values, see the *VMS DCL Concepts Manual*.

/BY_OWNER[=uic]

Selects only those files whose owner user identification code (UIC) matches the specified owner UIC. The default UIC is that of the current process.

Specify the UIC by using standard UIC format as described in the *VMS DCL Concepts Manual*.

/CONCATENATE (default)
/NOCONCATENATE

Creates one output file from multiple input files when you do not use wildcard characters in the output file specification. The /NOCONCATENATE qualifier generates multiple output files. A wildcard character in an input file specification results in a single output file consisting of the concatenation of all input files matching the file specification.

Files from Files–11 On-Disk Structure Level 2 disks are concatenated in alphanumeric order; if you specify a wildcard in the file version field, files are copied in descending order by version number. Files from Files–11 On-Disk Structure Level 1 disks are concatenated in random order.

/CONFIRM
/NOCONFIRM (default)

Controls whether a request is issued before each copy operation to confirm that the operation should be performed on that file. The following

responses are valid:

YES	NO	QUIT
TRUE	FALSE	Ctrl/Z
1	0	ALL
Return		

You can use any combination of uppercase and lowercase letters for word responses. You can abbreviate word responses to one or more letters (for example, T, TR, or TRU for TRUE), but these abbreviations must be unique. Affirmative answers are YES, TRUE, and 1. Negative answers include: NO, FALSE, 0, and pressing the Return key. Entering QUIT or pressing Ctrl/Z indicates that you want to stop processing the command at that point. When you respond by entering ALL, the command continues to process but no further prompts are given. If you type a response other than one of those in the list, DCL issues an error message and redisplays the prompt.

/CONTIGUOUS
/NOCONTIGUOUS
Specifies that the output file must occupy contiguous physical disk blocks. By default, the COPY command creates an output file in the same format as the corresponding input file. Also, by default, if not enough space exists for a contiguous allocation, the COPY command does not report an error. If you copy multiple input files of different formats, the output file may or may not be contiguous. You can use the /CONTIGUOUS qualifier to ensure that files are copied contiguously.

The /CONTIGUOUS qualifier has no effect when you copy files to or from tapes because the size of the file on tape cannot be determined until after it is copied to the disk. If you copy a file from a tape and want the file to be contiguous, use the COPY command twice: once to copy the file from the tape, and a second time to create a contiguous file.

/CREATED (default)
Modifies the time value specified with the /BEFORE or the /SINCE qualifier. The /CREATED qualifier selects files based on their dates of creation. This qualifier is incompatible with the /BACKUP, /EXPIRED, and /MODIFIED qualifiers, which also allow you to select files according to time attributes. If you specify none of these four time qualifiers, the default is the /CREATED qualifier.

/EXCLUDE=(filespec[,...])
Excludes the specified files from the copy operation. You can include a directory but not a device in the file specification. Wildcard characters (* and %) are allowed in the file specification. However, you cannot use relative version numbers to exclude a specific version. If you specify only one file, you can omit the parentheses.

/EXPIRED
Modifies the time value specified with the /BEFORE or the /SINCE qualifier. The /EXPIRED qualifier selects files according to their expiration dates. (The expiration date is set with the SET FILE/EXPIRATION_DATE command.) The /EXPIRED qualifier is incompatible with the /BACKUP, /CREATED, and /MODIFIED qualifiers, which also allow you to select files according to time attributes. If you specify none of these four time qualifiers, the default is the /CREATED qualifier.

/EXTENSION=n

Specifies the number of blocks to be added to the output file each time the file is extended. If you do not specify the /EXTENSION qualifier, the extension attribute of the corresponding input file determines the default extension attribute of the output file.

/LOG
/NOLOG (default)

Controls whether the COPY command displays the file specifications of each file copied.

When you use the /LOG qualifier, the COPY command displays the following for each copy operation:

* The file specifications of the input and output files

* The number of blocks or the number of records copied (depending on whether the file is copied on a block-by-block or record-by-record basis)

* The total number of new files created

/MODIFIED

Modifies the time value specified with the /BEFORE or the /SINCE qualifier. The /MODIFIED qualifier selects files according to the dates on which they were last modified. This qualifier is incompatible with the /BACKUP, /CREATED, and /EXPIRED qualifiers, which also allow you to select files according to time attributes. If you specify none of these four time modifiers, the default is the /CREATED qualifier.

/OVERLAY
/NOOVERLAY (default)

Requests that data in the input file be copied into the existing specified file, overlaying the existing data, rather than allocating new space for the file. The physical location of the file on disk does not change.

The /OVERLAY qualifier is ignored if the output file is written to a non-file-structured device.

/PROTECTION=(ownership[:access][,...])

Specifies protection for the output file. Specify the *ownership* parameter as system (S), owner (O), group (G), or world (W) and the *access* parameter as read (R), write (W), execute (E), or delete (D). The default protection, including any protection attributes not specified, is that of the existing output file. If no output file exists, the current default protection applies.

For more information on specifying protection codes, see the *VMS DCL Concepts Manual*.

/READ_CHECK
/NOREAD_CHECK (default)

Reads each record in the input files twice to verify that it has been read correctly.

/REPLACE
/NOREPLACE (default)

Requests that, if a file exists with the same file specification as that entered for the output file, the existing file is to be deleted. The COPY command allocates new space for the output file. In general, when you use the /REPLACE qualifier, include version numbers with the file specifications. By default, the COPY command creates a new version of a file if a file with that specification exists, incrementing the version number. The /NOREPLACE qualifier signals an error when a conflict in version numbers occurs.

/SINCE[=time]

Selects only those files dated after the specified time. You can specify time as an absolute time, as combination of absolute and delta times, or as one of the following keywords: TODAY (default), TOMORROW, or YESTERDAY. Specify one of the following qualifiers with the /SINCE qualifier to indicate the time attribute to be used as the basis for selection: /BACKUP, /CREATED (default), /EXPIRED, or /MODIFIED.

For complete information on specifying time values, see the *VMS DCL Concepts Manual*.

/TRUNCATE
/NOTRUNCATE (default)

Controls whether the COPY command truncates an output file at the end-of-file (EOF) when copying it. By default, the allocation of the input file determines the size of the output file.

/VOLUME=n

Places the output file on the specified relative volume number of a multivolume set. By default, the COPY command places the output file arbitrarily in a multivolume set.

/WRITE_CHECK
/NOWRITE_CHECK (default)

Reads each record in the output file after it was written to verify that the record was copied successfully and that the file can be read subsequently without error.

EXAMPLES

■ $ COPY TEST.DAT NEWTEST.DAT

In this example, the COPY command copies the contents of the file TEST.DAT from the default disk and directory to a file named NEWTEST.DAT on the same disk and directory. If a file named NEWTEST.DAT exists, the COPY command creates a new version of the file.

```
2   $ COPY ALPHA.TXT TMP
    $ COPY ALPHA.TXT .TMP
```

> In this example, the first COPY command copies the file ALPHA.TXT into a file named TMP.TXT. The COPY command uses the file type of the input file to complete the file specification for the output file. The second COPY command creates a file named ALPHA.TMP. The COPY command uses the file name of the input file to name the output file.

```
3   $ COPY/LOG TEST.DAT  NEW.DAT;1/REPLACE
    %COPY-I-REPLACED, DBA0:[MAL]NEW.DAT;1 being replaced
    %COPY-S-COPIED, DBA0:[MAL]TEST.DAT;1 copied to DBA0:[MAL]NEW.DAT;1 (1 block)
```

> In this example, the /REPLACE qualifier requests that the COPY command replace an existing version of the output file with the new file. The first message from the COPY command indicates that it is replacing an existing file. The version number in the output file must be explicit; otherwise, the COPY command creates a new version of the file NEW.DAT.

```
4   $ COPY *.COM [MALCOLM.TESTFILES]
```

> In this example, the COPY command copies the highest versions of files in the current default directory with the file type COM to the subdirectory MALCOLM.TESTFILES.

```
5   $ COPY/LOG *.TXT *.OLD
    %COPY-S-COPIED, DBA0:[MAL]A.TXT;2 copied to DBA0:[MAL]A.OLD;2 (1 block)
    %COPY-S-COPIED, DBA0:[MAL]B.TXT;2 copied to DBA0:[MAL]B.OLD;2 (1 block)
    %COPY-S-COPIED, DBA0:[MAL]G.TXT;2 copied to DBA0:[MAL]G.OLD;2 (4 blocks)
    %COPY-S-NEWFILES, 3 files created
```

> In this example, the COPY command copies the highest versions of files with file types of TXT into new files. Each new file has the same file name as an existing file, but a file type of OLD. The last message from the COPY command indicates the number of new files that have been created.

```
6   $ COPY/LOG  A.DAT,B.MEM C.*
    %COPY-S-COPIED, DBA0:[MAL]A.DAT;5 copied to DBA0:[MAL]C.DAT;11 (1 block)
    %COPY-S-COPIED, DBA0:[MAL]B.MEM;2 copied to DBA0:[MAL]C.MEM;24 (58 records)
    %COPY-S-NEWFILES, 2 files created
```

> In this example, the two input file specifications are separated with a comma. The asterisk wildcard character in the output file specification indicates that two output files are to be created. For each copy operation, the COPY command uses the file type of the input file to name the output file.

7 $ COPY/LOG *.TXT TXT.SAV
%COPY-S-COPIED, DBA0:[MAL]A.TXT;2 copied to DBA0:[MAL]TXT.SAV;1 (1 block)
%COPY-S-APPENDED, DBA0:[MAL]B.TXT;2 appended to DBA0:[MAL]TXT.SAV;1 (3 records)
%COPY-S-APPENDED, DBA0:[MAL]G.TXT;2 appended to DBA0:[MAL]TXT.SAV;1 (51 records)
%COPY-S-NEWFILES, 1 file created

In this example, the COPY command copies the highest versions of all files with the file type TXT to a single output file named TXT.SAV. After the first input file is copied, the messages from the COPY command indicate that subsequent files are being appended to the output file.

Note that, if you use the /NOCONCATENATE qualifier in this example, the COPY command creates one TXT.SAV file for each input file. Each TXT.SAV file has a different version number.

8 $ COPY MASTER.DOC DBA1:[BACKUP]

In this example, the COPY command copies the highest version of the file MASTER.DOC to the device DBA1. If no file named MASTER.DOC exists in the directory [BACKUP], the COPY command assigns the version number of the input file to the output file. You must have write (W) access to the directory [BACKUP] on device DBA1 for the command to work.

9 $ COPY SAMPLE.EXE DALLAS::DISK2:[000,000]SAMPLE.EXE/CONTIGUOUS

In this example, the COPY command copies the file SAMPLE.EXE on the local node to a file with the same name at remote node DALLAS. The /CONTIGUOUS qualifier indicates that the output file is to occupy consecutive physical disk blocks. You must have write (W) access to the device DISK2 on remote node DALLAS for the command to work.

10 $ COPY *.* PRTLND::*.*

In this example, the COPY command copies all files within the user directory at the local node to the remote node PRTLND. The new files have the same names as the input file. You must have write (W) access to the default directory on remote node PRTLND for the command to work.

11 $ COPY BOSTON::DISK2:TEST.DAT;5
_To: DALLAS"SAM SECReturn"::DISK0:[MODEL.TEST]TEST.DAT/ALLOCATION=50

In this example, the COPY command copies the file TEST.DAT;5 on the device DISK2 at node BOSTON to a new file named TEST.DAT at remote node DALLAS. The /ALLOCATION qualifier initially allocates 50 blocks for the new file TEST.DAT at node DALLAS. The access control string SAM SECReturn is used to access the remote directory.

```
12   $ MOUNT   TAPED1:   VOL025   TAPE:
     $ COPY  TAPE:*.*   *
```

> In this example, the MOUNT command requests that the volume labeled VOL025 be mounted on the magnetic tape device TAPED1 and assigns the logical name TAPE to the device.
>
> The COPY command uses the logical name TAPE as the input file specification, requesting that all files on the magnetic tape be copied to the current default disk and directory. All the files copied retain their file names and file types.

```
13   $ ALLOCATE CR:
     _CR1: ALLOCATED
     $ COPY CR1: CARDS.DAT
     $ DEALLOCATE CR1:
```

> In this example, the ALLOCATE command allocates a card reader for exclusive use by the process. The response from the ALLOCATE command indicates the device name of the card reader, CR1.
>
> After the card reader is allocated, you can place a deck of cards in the reader and enter the COPY command, specifying the card reader as the input file. The COPY command reads the cards into the file CARDS.DAT. The end-of-file (EOF) in the card deck must be indicated with an EOF card (12-11-0-1-6-7-8-9 overpunch).
>
> The DEALLOCATE command relinquishes use of the card reader.

CREATE

Creates a sequential disk file (or files).

FORMAT

CREATE *filespec[,...]*

PARAMETER

filespec[,...]
Specifies the name of one or more input files to be created. Wildcard characters are not allowed. If you omit either the file name or the file type, the CREATE command does not supply any defaults. The file name or file type is null. If the specified file already exists, a new version is created.

DESCRIPTION

The CREATE command creates a new sequential disk file. In interactive mode, each separate line that you enter after you enter the command line becomes a record in the newly created file. To terminate the file input, press Ctrl/Z.

When you enter the CREATE command from a command procedure file, the system reads all subsequent records in the command procedure file into the new file until it encounters a dollar sign ($) in the first position in a record. Terminate the file input with a line with a dollar sign in column 1 (or with the end of the command procedure).

If you use an existing file specification with the CREATE command, the newly created file has a higher version number than any existing files with the same specification.

If you use the CREATE command to create a file in a logical name search list, the file will only be created in the first directory produced by the logical name translation.

Normally, the owner of the output file will be the same as the creator of the output file. However, if a user with extended privileges creates the output file, the owner will be the owner of the parent directory or any previous versions of the output file.

Extended privileges include any of the following:

- SYSPRV (system privilege) or BYPASS
- System user identification code (UIC)
- GRPPRV (group privilege) if the owner of the parent directory (or previous version of the output file) is in the same group as the creator of the new output file
- An identifier (with the resource attribute) representing the owner of the parent directory (or previous version of the output file)

QUALIFIERS	/LOG

/LOG
/NOLOG (default)
Displays the file specification of each new file created as the command executes.

/OWNER_UIC=uic
Requires SYSPRV (system privilege) privilege to specify a user identification code (UIC) other than your own.

Specifies the UIC to be associated with the file being created. Specify the UIC by using standard UIC format as described in the *VMS DCL Concepts Manual*.

/PROTECTION=(ownership[:access][,...])
Specifies protection for the file. Specify the *ownership* parameter as system (S), owner (O), group (G), or world (W) and the *access* parameter as read (R), write (W), execute (E), or delete (D). If you do not specify a value for each access category, or if you omit the /PROTECTION qualifier, the CREATE command applies the following protection for each unspecified category:

File Already Exists?	Protection Applied
Yes	Protection of the existing file
No	Current default protection

For more information on specifying protection codes, see the *VMS DCL Concepts Manual*.

/VOLUME=n
Places the file on the specified relative volume of a multivolume set. By default, the file is placed arbitrarily in a multivolume set.

EXAMPLES

1
```
$ CREATE MEET.TXT
John, Residents in the apartment complex will hold their annual meeting
this evening.  We hope to see you there, Regards, Elwood
Ctrl/Z
```

The CREATE command in this example creates a text file named MEET.TXT in your default directory. The text file MEET.TXT contains the lines that follow until the Ctrl/Z.

CREATE

2
```
$  CREATE A.DAT, B.DAT
Input line one for A.DAT...
Input line two for A.DAT...
      .
      .
      .

  Ctrl/Z

Input line one for B.DAT...
Input line two for B.DAT...
      .
      .
      .

  Ctrl/Z
$
```

After you enter the CREATE command from the terminal, the system reads input lines into the sequential file A.DAT until Ctrl/Z terminates the first input. The next set of input data is placed in the second file, B.DAT. Again, Ctrl/Z terminates the input.

3
```
$ FILE = F$SEARCH("MEET.TXT")
$ IF FILE .EQS. ""
$ THEN CREATE MEET.TXT
  John, Residents in the apartment complex will hold their annual meeting
  this evening.  We hope to see you there, Regards, Elwood
$ ELSE TYPE MEET.TXT
$ ENDIF
$ EXIT
```

In this example, the command procedure searches the default disk and directory for the file MEET.TXT. If the command procedure determines that the file does not exist it creates a file named MEET.TXT using the CREATE command.

4

...Input Data...

$ RUN WEATHER

$ LINK WEATHER

$ FORTRAN WEATHER

...Source Statements...

$ CREATE WEATHER.FOR

$ PASSWORD HENRY

$ JOB HIGGINS
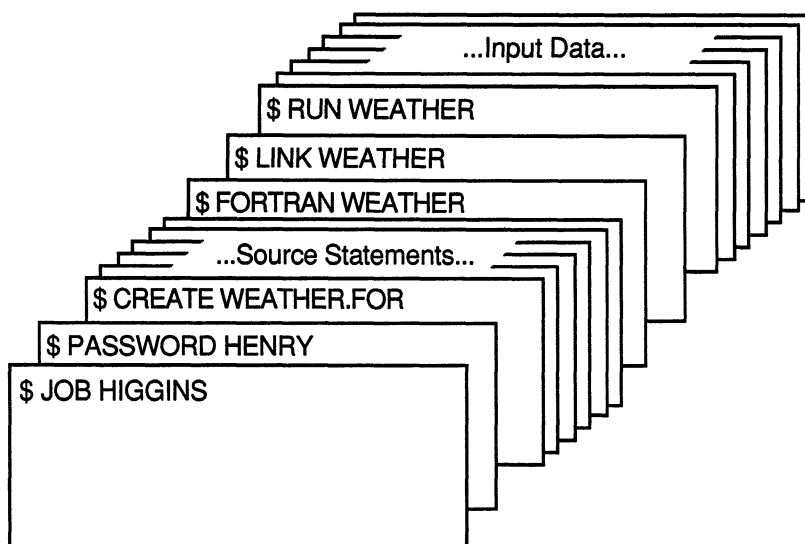
ZK-0781-GE

In this batch job example, the CREATE command creates a FORTRAN
source file WEATHER.FOR. Records are read into that file until the
system encounters a dollar sign in the first position of the record
$ FORTRAN WEATHER. The next commands compile, link, and run
the file just created. Input data follows the RUN command.

CREATE

```
                                                              $ EOJ
                                                        $ @WEATHER
End of Input Stream ──────────────────────────────→   $ EOD
Input Stream for ─────────────────────────────→      $ RUN WEATHER
CREATE Command
                                                   $ LINK WEATHER
                                              $ FORTRAN WEATHER
Input Stream with ──────────────────→     $ DECK
Dollar Signs Follows
                                       $ CREATE WEATHER.COM
                                 $ PASSWORD HENRY
                          $ JOB HIGGINS
```
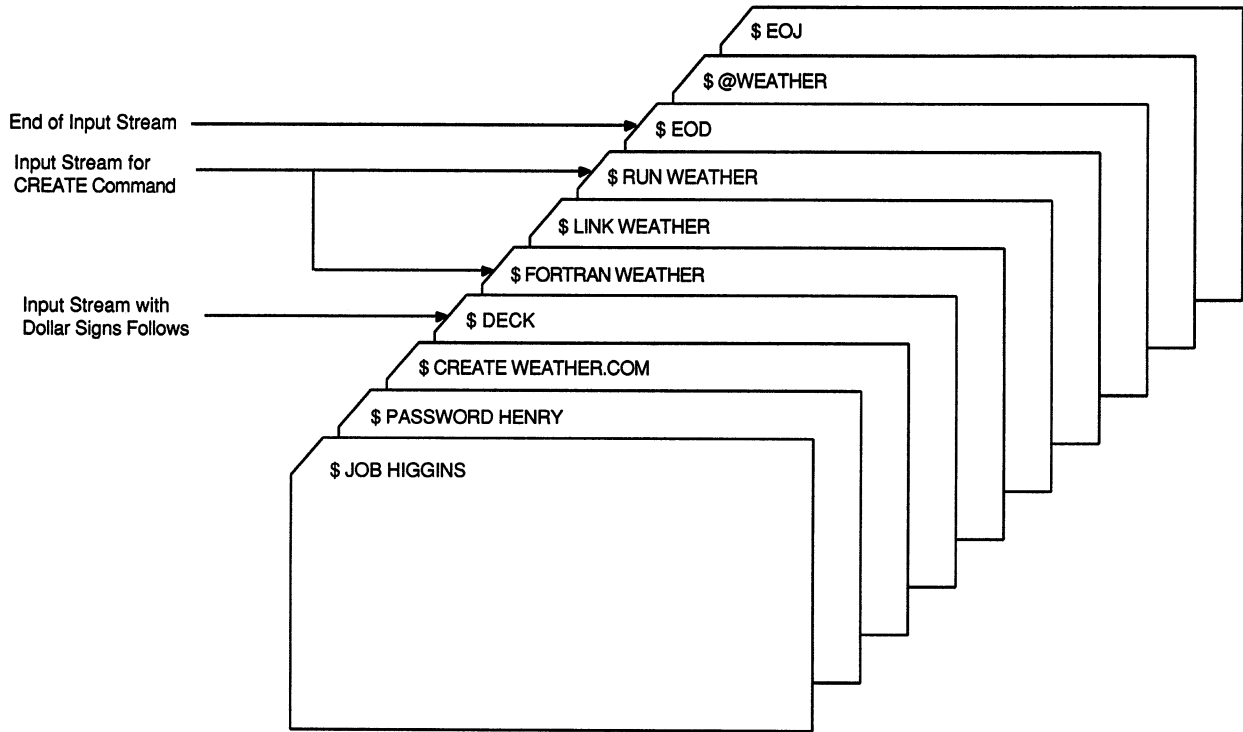
ZK-0782-GE

This batch job example uses the CREATE command to create a command procedure from data in the input stream. The DECK command is required so that subsequent lines that begin with a dollar sign are not executed as commands, but are accepted as input records. The EOD command signals the end-of-file (EOF) for the data records. Then the WEATHER procedure is executed with the @ (execute procedure) command.

CREATE/DIRECTORY

Creates one or more new directories or subdirectories. The /DIRECTORY qualifier is required.

Requires write (W) access to the master file directory (MFD) to create a first-level directory. On a system volume, generally only users with a system user identification code (UIC) or the SYSPRV (system privilege) or BYPASS user privileges have write (W) access to the MFD to create a first-level directory.

Requires write (W) access to the lowest level directory that currently exists to create a subdirectory.

FORMAT **CREATE/DIRECTORY** *directory-spec[,...]*

PARAMETER *directory-spec[,...]*
Specifies the name of one or more directories or subdirectories to be created. The directory specification optionally can be preceded by a device name (and colon [:]). The default is the current default directory. Wildcard characters are not allowed. When you create a subdirectory, separate the names of the directory levels with periods (.).

Note that it is possible to create a series of nested subdirectories with a single CREATE/DIRECTORY command. For example, [a.b.c] can be created, even though neither [a.b] nor [a] exists at the time the command is entered. Each subdirectory will be created, starting with the highest level and proceeding downward.

DESCRIPTION The CREATE/DIRECTORY command creates new directories as well as subdirectories. Special privileges are needed to create new first-level directories. (See the restrictions noted above.) Generally, users have sufficient privileges to create subdirectories in their own directories. Use the SET DEFAULT command to move from one directory to another.

QUALIFIERS */LOG*
/NOLOG (default)
Controls whether the CREATE/DIRECTORY command displays the directory specification of each directory after creating it.

/OWNER_UIC[=option]
Requires SYSPRV (system privilege) privilege for a user identification code (UIC) other than your own.

Specifies the owner UIC for the directory. The default is your UIC. You can specify the keyword PARENT in place of a UIC to mean the UIC of the parent (next-higher-level) directory. If a user with privileges creates a subdirectory, by default, the owner of the subdirectory will be the owner of the parent directory (or the owner of the MFD, if creating a main level

directory). If you do not specifiy the /OWNER_UIC qualifier when creating a directory, the command assigns ownership as follows: (1) if you specify the directory name in either alphanumeric or subdirectory format, the default is your UIC (unless you are privileged in which case the UIC defaults to the parent directory); (2) if you specify the directory in UIC format, the default is the specified UIC.

Specify the UIC by using standard UIC format as described in the *VMS DCL Concepts Manual*.

/PROTECTION=(ownership[:access][,...])

Specifies protection for the directory. Specify the *ownership* parameter as system (S), owner (O), group (G), or world (W) and the *access* parameter as read (R), write (W), execute (E), or delete (D). The default protection is the protection of the parent directory (the next-higher level directory, or the master directory for top-level directories) minus any delete (D) access.

If you are creating a first-level directory, then the next-higher-level directory is the MFD. (The protection of the MFD is established by the INITIALIZE command.)

For more information on specifying protection code, see the *VMS DCL Concepts Manual*.

/VERSION_LIMIT=n

Specifies the number of versions of any one file that can exist in the directory. If you exceed the limit, the system deletes the lowest numbered version. A specification of 0 means no limit. The maximum number of versions allowed is 32,767. The default is the limit for the parent (next-higher-level) directory.

When you change the version limit setting, the new limit applies only to files created after the setting was changed. New versions of files created before the change are subject to the previous version limit.

/VOLUME=n

Requests that the directory file be placed on the specified relative volume of a multivolume set. By default, the file is placed arbitrarily within the multivolume set.

EXAMPLES

1　$ CREATE/DIRECTORY/VERSION_LIMIT=2 $DISK1:[ACCOUNTS.MEMOS]

In this example, the CREATE/DIRECTORY command creates a subdirectory named MEMOS in the ACCOUNTS directory on $DISK1. No more than two versions of each file can exist in the directory.

2　$ CREATE/DIRECTORY/PROTECTION=(SYSTEM:RWED,OWNER:RWED,GROUP,WORLD) -
_$ [MALCOLM.SUB.HLP]

In this example, the CREATE/DIRECTORY command creates a subdirectory named [MALCOLM.SUB.HLP]. The protection on the subdirectory allows read (R), write (W), execute (E), and delete (D) access for the system and owner categories, but prohibits all access for the group or world categories.

3    $ `CREATE/DIRECTORY DISK2:[MALCOLM]`

> In this example, the CREATE/DIRECTORY command creates a directory named [MALCOLM] on the device DISK2. Special privileges are required to create a first-level directory.

4    $ `CREATE/DIRECTORY [MALCOLM.SUB]`
      $ `SET DEFAULT [MALCOLM.SUB]`

> In this example, the CREATE/DIRECTORY command creates a subdirectory named [MALCOLM.SUB]. This directory file is placed in the directory named [MALCOLM]. The command SET DEFAULT [MALCOLM.SUB] changes the current default directory to this subdirectory. All files subsequently created are cataloged in [MALCOLM.SUB].

5    $ `CREATE/DIRECTORY [FRED.SUB1.SUB2.SUB3]`

> In this example, the CREATE/DIRECTORY command creates a top-level directory ([FRED]) and three subdirectories ([FRED.SUB1], [FRED.SUB1.SUB2], and [FRED.SUB1.SUB2.SUB3]).

CREATE/FDL

Invokes the Create/FDL Utility, which uses the specifications in a File Definition Language (FDL) file to create a new, empty data file. The /FDL qualifier is required. For a complete description of the Create/FDL Utility, see the *VMS File Definition Language Facility Manual*.

FORMAT   **CREATE/FDL**=*fdl-filespec [filespec]*

CREATE/NAME_TABLE

Creates a new logical name table. The /NAME_TABLE qualifier is required.

FORMAT   **CREATE/NAME_TABLE** *table-name*

PARAMETER   *table-name*
Specifies a string of 1 to 31 characters that identifies the logical name table you are creating. The string can include alphanumeric characters, the dollar sign ($), and the underscore (_). This name is entered as a logical name in either the process directory logical name table (LNM$PROCESS_DIRECTORY) or the system directory logical name table (LNM$SYSTEM_DIRECTORY).

DESCRIPTION   The CREATE/NAME_TABLE command creates a new logical name table. The name of the table is contained within the LNM$PROCESS_ DIRECTORY directory table if the table is process-private, and within the LNM$SYSTEM_DIRECTORY directory table if the table is shareable.

Every new table has a parent table, which determines whether the new table is process-private or shareable. To create a process-private table, use the /PARENT_TABLE qualifier to specify the name of a process-private table (the process directory table). To create a shareable table, specify the parent as a shareable table.

If you do not explicitly provide a parent table, the CREATE/NAME_ TABLE command creates a process-private table whose parent is LNM$PROCESS_DIRECTORY; that is, the name of the table is entered in the process directory.

Every table has a size quota. The quota may either constrain the potential growth of the table or indicate that the table's size can be virtually unlimited. The description of the /QUOTA qualifier explains how to specify a quota.

To specify an access mode for the table you are creating, use the /USER_ MODE, the /SUPERVISOR_MODE, or the /EXECUTIVE_MODE qualifier. If you specify more than one of these qualifiers, only the last one entered is accepted. If you do not specify an access mode, then a supervisor-mode table is created.

To delete a logical name table, use the DEASSIGN command, specify the name of the table you want to delete, and use the /TABLE qualifier to specify the directory table where the name of the table was entered.

QUALIFIERS	**/ATTRIBUTES[=(keyword[,...])]**

Specifies attributes for the logical name table. If you specify only one keyword, you can omit the parentheses. If you do not specify the /ATTRIBUTES qualifier, no attributes are set.

You can specify the following keywords for attributes:

CONFINE
: Does not copy the table name or the logical names contained in the table into a spawned subprocess; used only when creating a private logical name table. If a table is created with the CONFINE attribute, all names subsequently entered into the table are also confined.

NO_ALIAS
: No identical names (either logical names or names of logical name tables) may be created in an outer (less privileged) mode in the current directory. If you do not specify the NO_ALIAS attribute, then the table may be "aliased" by an identical name created in an outer access mode. Deletes any previously created identical table names in an outer access mode in the same logical name table directory.

SUPERSEDE
: Creates a new table that supersedes any previous (existing) table that contains the name, access mode, and directory table that you specify. The new table is created regardless of whether the previous table exists. (If you do not specify the SUPERSEDE attribute, the new table is not created if the previous table exists.)

: If you specify or accept the default for the qualifier /LOG, you receive a message indicating the result.

/EXECUTIVE_MODE
Requires SYSNAM (system logical name) privilege.

Creates an executive-mode logical name table. If you specify executive mode, but do not have SYSNAM privilege, a supervisor-mode logical name table is created.

/LOG (default)
/NOLOG

Controls whether an informational message is generated when the SUPERSEDE attribute is specified, or when the table already exists but the SUPERSEDE attribute is not specified. The default is the /LOG qualifier; that is, the informational message is displayed.

/PARENT_TABLE=table
Requires execute (E) access to the parent table and SYSPRV (system privilege) privilege to create a shareable logical name table.

Specifies the name of the parent table. The parent table determines whether a table is private or shareable; it also determines the size quota of the table. If you do not specify a parent table, the default table is LNM$PROCESS_DIRECTORY. A shareable table has LNM$SYSTEM_DIRECTORY as its parent table. The parent table must have the same access mode or a higher level access mode than the one you are creating.

/PROTECTION=(ownership[:access][,...])
Applies the specified protection to shareable name tables. Specify the *ownership* parameter as system (S), owner (O), group (G), or world (W) and the *access* parameter as read (R), write (W), execute (E), or delete (D). The default protection is (S:RWED, O:RWED, G, W).

For more information on specifying protection codes, see the *VMS DCL Concepts Manual*.

The /PROTECTION qualifier affects only shareable logical name tables; it does not affect process-private logical name tables.

/QUOTA=number-of-bytes

Specifies the size limit of the logical name table. The size of each logical name entered in the new table is deducted from this size limit. The new table's quota is statically subtracted from the parent table's quota holder. The parent table's quota holder is the first logical name table encountered when working upward in the table hierarchy that has an explicit quota and is therefore its own quota holder. If the /QUOTA qualifier is not specified or the size limit is 0, the parent table's quota holder becomes the new table's quota holder and space is dynamically withdrawn from it whenever a logical name is entered in this new table. If you do not specify the /QUOTA qualifier, or if you specify /QUOTA=0, the table has unlimited quota.

/SUPERVISOR_MODE (default)

Creates a supervisor-mode logical name table. If you do not specify a mode, a supervisor-mode logical name table is created.

/USER_MODE

Creates a user-mode logical name table. If you do not explicitly specify a mode, a supervisor-mode logical name table is created.

EXAMPLES

1
```
$  CREATE/NAME_TABLE TEST_TAB
$  SHOW LOGICAL TEST_TAB
%SHOW-S-NOTRAN, no translation for logical name TEST_TAB
$  SHOW LOGICAL/TABLE=LNM$PROCESS_DIRECTORY TEST_TAB
```

In this example, the CREATE/NAME_TABLE command creates a new table called TEST_TAB. By default, the name of the table is entered in the process directory. The first SHOW LOGICAL command does not find the name TEST_TAB because it does not, by default, search the process directory table. You must use the /TABLE qualifier to request that the process directory be searched.

2
```
$  CREATE/NAME_TABLE/ATTRIBUTES=CONFINE EXTRA
$  DEFINE/TABLE=EXTRA MYDISK DISK4:
$  DEFINE/TABLE=LNM$PROCESS_DIRECTORY LNM$FILE_DEV -
_$  EXTRA, LNM$PROCESS, LNM$JOB, LNM$GROUP, LNM$SYSTEM
$  TYPE MYDISK:[COHEN]EXAMPLE1.LIS
```

This example creates a new logical name table called EXTRA that is created with the CONFINE attribute. Therefore, the EXTRA table and the names it contains will not be copied to subprocesses.

Next, the logical name MYDISK is placed into the table EXTRA. To use the name MYDISK in file specifications, you must make sure that the table EXTRA is searched when RMS parses file specifications. To do this, you can define a process-private version of the logical name LNM$FILE_ DEV to include the name EXTRA as one of its equivalence strings. (The system uses LNM$FILE_DEV to determine the tables to search during logical name translation for device or file specifications, and will use the process-private version of the logical name before using the default system version.) After you define LNM$FILE_DEV, the system searches the following tables during logical name translation: EXTRA, your process table, your job table, your group table, and the system table. Now, you can use the name MYDISK in a file specification and the equivalence string DISK4 will be substituted.

CREATE/TERMINAL

The DCL command CREATE/TERMINAL creates a window that emulates another terminal type.

Note: **At present, only DECterm windows are available with this command.**

FORMAT

CREATE/TERMINAL *[command-string]*

PARAMETER

command-string

Specifies a command string that is to be executed in the context of the created subprocess. You cannot specify this parameter with the /DETACH or the /NOPROCESS qualifier. The CREATE/TERMINAL command is used in much the same way as the SPAWN command.

DESCRIPTION

The CREATE/TERMINAL command creates a subprocess of your current process. When the subprocess is created, the process-permanent open files and any image or procedure context are *not* copied from the parent process. The subprocess is set to command level 0 (DCL level with the current prompt).

If you do not specify the /PROCESS qualifier, the name of this subprocess is composed of the same base name as the parent process and a unique number. For example, if the parent process name is SMITH, the subprocess name can be SMITH_1, SMITH_2, and so on.

The LOGIN.COM file of the parent process is not executed for the subprocess, because the context is copied separately, allowing quicker initialization of the subprocess. When the /WAIT qualifier is in effect, the parent process remains in hibernation until the subprocess terminates and returns control to the parent by using the ATTACH command.

You should use the LOGOUT command to terminate the subprocess and return to the parent process. You can also use the ATTACH command to transfer control of the terminal to another process in the subprocess tree, including the parent process. (The SHOW PROCESS/SUBPROCESS command displays the process in the subprocess tree and points to the current process.)

Note: **Because a tree of subprocesses can be established using the CREATE/TERMINAL command, you must be careful when terminating any process in the tree. When a process is terminated, all the subprocesses below that point in the tree are automatically terminated. For example, the SPAWN/NOWAIT CREATE /TERMINAL command creates a subprocess that exits as soon as the DECterm window is created. Once this process exits, the DECterm window disappears. Instead, use the SPAWN/NOWAIT CREATE/TERMINAL/WAIT command to allow the process to continue.**

Qualifiers with the CREATE/TERMINAL command must directly follow the command verb. The *command-string* parameter begins after the last qualifier and continues to the end of the command line.

QUALIFIERS

/APPLICATION_KEYPAD
Sets the APPLICATION_KEYPAD terminal characteristic in the created terminal window. If the /APPLICATION_KEYPAD or the /NUMERIC_KEYPAD qualifier is not specified, the default is to inherit the characteristic from the parent. (See also /NUMERIC_KEYPAD.)

/BIG_FONT
Specifies that the big font (as specified in resource files) be selected when the created terminal window is initialized. It is an error to specify the /BIG_FONT qualifier in combination with the /LITTLE_FONT qualifier. If you do not specify either the /BIG_FONT or the /LITTLE_FONT qualifier, the initial font is the big font.

/BROADCAST
/NOBROADCAST
Determines whether the terminal window is created with broadcast messages enabled. If neither qualifier is specified, the created terminal window inherits the broadcast characteristic of the parent.

/CARRIAGE_CONTROL
/NOCARRIAGE_CONTROL
Determines whether carriage-return and line-feed characters are prefixed to the subprocess's prompt string. By default, the CREATE/TERMINAL command copies the current setting of the parent process. The CARRIAGE_CONTROL qualifier is used only with the /NODETACH qualifier.

/CLI=cli-file-spec
/NOCLI
Specifies the name of a command language interpreter (CLI) to be used by the subprocess. The default CLI is the same as that of the parent process (defined in SYSUAF). If you specify the /CLI qualifier, the attributes of the parent process are copied to the subprocess. The CLI you specify must be located in SYS$SYSTEM and have the file type EXE. This qualifier is used only with the /NODETACH qualifier.

/CONTROLLER=filespec
Specifies the name of the terminal window controller image. This name allows the CREATE/TERMINAL command to create a window on a variant controller, such as for a language not supported by the base product. For a DECterm window, the default is SYS$SYSTEM:DECW$TERMINAL.EXE. The device and directory default to SYS$SYSTEM and the file type defaults to EXE.

Note: **The "name" field of the file name as returned by $PARSE is used to form the mailbox logical name. For example, if the file "name" is DECW$TERMINAL, the mailbox logical name will be DECW$TERMINAL_MAILBOX_node::0.0. For backward compatibility, the controller also defines a logical**

name DECW$DECTERM_MAILBOX_host::0.0 to point to the same mailbox.

/DEFINE_LOGICAL=({logname, TABLE=tablename} [,...])

Specifies one or more logical names that are set to the name of the created pseudo-terminal device. Each element in the list is either a logical name or TABLE= followed by the name of a logical name table in which all subsequent logical names will be entered. The default is the process logical name table.

/DETACH
/NODETACH (default)

Determines whether the created terminal process is detached or a subprocess of the current process. The /DETACH qualifier cannot be used with the *command-string* parameter.

/DISPLAY=display-name

Specifies the name of the display on which to create the terminal window. If this parameter is omitted, the DECW$DISPLAY logical name is used.

/ESCAPE
/NOESCAPE

Sets or clears the ESCAPE characteristic of the created terminal window. The default is to inherit the characteristic of the parent.

/FALLBACK
/NOFALLBACK

Sets or clears the FALLBACK characteristic of the created terminal window. The default is to inherit the characteristic of the parent.

/HOSTSYNC (default)
/NOHOSTSYNC

Sets or clears the HOSTSYNC characteristic of the created terminal window. The default is to inherit the characteristic of the parent.

/INPUT=filespec

Specifies an alternate input file or device to use as SYS$INPUT for the new process. The default is to use the created terminal window for input. This qualifier can be used with or without the /DETACH qualifier.

/INSERT

Creates the terminal window with insert mode as the default for line editing. If the /INSERT or the /OVERSTRIKE qualifier is not specified, the default is to inherit the characteristic from the parent. (See also /OVERSTRIKE.)

/KEYPAD (default)
/NOKEYPAD

Determines whether keypad definitions and the current keypad state are copied from the parent process. This qualifier is used only with the /NODETACH qualifier.

/LINE_EDITING
/NOLINE_EDITING

Determines whether the terminal window is created with line editing enabled. If neither qualifier is specified, the created terminal window inherits the line editing characteristic of the parent.

/LITTLE_FONT

Specifies that the little font (as specified in resource files) be selected when the created terminal window is initialized. It is an error to specify the /LITTLE_FONT qualifier in combination with the /BIG_FONT qualifier. If you do not specify either the /BIG_FONT or the /LITTLE_FONT qualifier, the initial font is the big font.

/LOGGED_IN (default)
/NOLOGGED_IN

Determines whether a prompt for a user name and password are supplied (/NOLOGGED_IN) or the created terminal window is logged in automatically (/LOGGED_IN). This qualifier is used only with the /DETACH qualifier.

/LOGICAL_NAMES (default)
/NOLOGICAL_NAMES

Determines whether the created terminal window inherits the parent's logical names. This qualifier is used only with the /NODETACH qualifier.

/NOTIFY
/NONOTIFY (default)

Determines whether a notification message is broadcast to the parent when the created terminal window exits. This qualifier is used only with the /NODETACH qualifier.

/NUMERIC_KEYPAD

Sets the NUMERIC_KEYPAD terminal characteristic in the created terminal window. If the /NUMERIC_KEYPAD or the /APPLICATION_ KEYPAD qualifier is not specified, the default is to inherit the characteristic from the parent. (See also /APPLICATION_KEYPAD.)

/OVERSTRIKE

Creates the terminal window with overstrike mode as the default for line editing. If the /OVERSTRIKE or the /INSERT qualifier is not specified, the default is to inherit the characteristic from the parent. (See also /INSERT.)

/PASTHRU
/NOPASTHRU

Sets or clears the PASTHRU characteristic in the created terminal window. The default is to inherit the characteristic of the parent.

/PROCESS (default)
/PROCESS=process-name
/NOPROCESS

Specifies the name of the process or subprocess to be created. The /NOPROCESS qualifier causes a window to be created without a process. If you specify the /PROCESS qualifier without a process name, a unique process name is assigned with the same base name as the parent process and a unique number. The default process name format is *username_n*. If you specify a process name that already exists, an error message is displayed. This qualifier is used with either the /DETACH or the /NODETACH qualifier.

/PROMPT=prompt

Specifies the prompt string of the created terminal window. This qualifier is used only with the /NODETACH qualifier.

/READSYNC
/NOREADSYNC

Sets or clears the READSYNC terminal characteristic in the created terminal window. The default is to inherit the characteristic from the parent.

/RESOURCE_FILE=filespec

Specifies that the created terminal window use the resource file "filespec" instead of the default resource file, DECW$USER_DEFAULTS:DECW$TERMINAL_DEFAULT.DAT.

/SYMBOLS (default)
/NOSYMBOLS

Determines whether the subprocess inherits the parent's DCL symbols. This qualifier is used only with the /NODETACH qualifier.

/TABLE=command-table

Specifies the name of an alternate command table to be used by the subprocess. This qualifier is used only with the /NODETACH qualifier.

/TTSYNC
/NOTTSYNC

Sets or clears the TTSYNC terminal characteristic in the created terminal window; the default is to inherit the characteristic of the parent.

/TYPE_AHEAD
/NOTYPE_AHEAD

Sets or clears the TYPE_AHEAD terminal characteristic in the created terminal window. The default is to inherit the characteristic of the parent.

/WAIT
/NOWAIT (default)

Requires that you wait for the subprocess to terminate before you enter another DCL command. The /NOWAIT qualifier allows you to enter new commands while the subprocess is running. This qualifier is used only with the /NODETACH qualifier.

/WINDOW_ATTRIBUTES=(parameter [,...])

Specifies initial attributes for the created terminal window to override the defaults read from the resource file. These parameters include:

Parameter	Description
BACKGROUND	The background color.
FOREGROUND	The foreground color.
WIDTH	The width, in pixels.
HEIGHT	The height, in pixels.
X_POSITION	The x-position, in pixels.
Y_POSITION	The y-position, in pixels.
ROWS	The number of rows in the window, in character cells. If the Auto Resize Window option is enabled, the ROWS and COLUMNS parameters override the size specified by the WIDTH and HEIGHT parameters.
COLUMNS	The number of columns in the window, in character cells. If the Auto Resize Window option is enabled, the ROWS and COLUMNS parameters override the size specified by the WIDTH and HEIGHT parameters.
INITIAL_STATE	The initial state of the window, either ICON or WINDOW.
TITLE	A character string specifying the window title.
ICON_NAME	A character string specifying the window icon name.
FONT	The name of the font to be used in the window. If you specify the /LITTLE_FONT qualifier, or omit both the /LITTLE_FONT and /BIG_FONT qualifiers, this overrides the name of the little font that is set in the resource files; otherwise it overrides the name of the big font. The font name can be a logical name, and it can be (but does not have to be) the base font in a complete font set.

EXAMPLE

```
$  CREATE/TERMINAL=DECTERM -
_$ /DISPLAY=MYNODE::0 -
_$ /WINDOW_ATTRIBUTES=( -
_$ ROWS=36, -
_$ COLUMNS=80,
_$ TITLE="REMOTE TERMINAL", -
_$ ICON_NAME="REMOTE TERMINAL" )
```

In this example, the command creates a detached process in a DECterm window on node MYNODE:: that is 36 rows by 80 columns and has its title and icon name set to "Remote terminal".

DEALLOCATE

Makes an allocated device available to other processes (but does not deassign any logical name associated with the device).

FORMAT

DEALLOCATE *device-name[:]*

PARAMETER

device-name[:]

Name of the device to be deallocated. The device name can be a physical device name or a logical name. On a physical device name, the controller defaults to A and the unit to 0. This parameter is incompatible with the /ALL qualifier.

QUALIFIER

/ALL

Deallocates all devices currently allocated by your process. This qualifier is incompatible with the *device-name* parameter.

EXAMPLES

1
```
$ DEALLOCATE  DMB1:
```

In this example, the DEALLOCATE command deallocates unit 1 of the RK06/RK07 devices on controller B.

2
```
$ ALLOCATE   MT:   TAPE
%DCL-I-ALLOC, _MTB1:  allocated
     .
     .
     .
$ DEALLOCATE  TAPE:
```

In this example, the ALLOCATE command requests that any magnetic tape drive be allocated and assigns the logical name TAPE to the device. The response to the ALLOCATE command indicates the successful allocation of the device MTB1. The DEALLOCATE command specifies the logical name TAPE to release the tape drive.

3
```
$ DEALLOCATE/ALL
```

In this example, the DEALLOCATE command deallocates all devices that are currently allocated.

DEASSIGN

Cancels a logical name assignment that was made with one of the following commands: ALLOCATE, ASSIGN, DEFINE, or MOUNT. The DEASSIGN command also deletes a logical name table that was created with the CREATE /NAME_TABLE command.

FORMAT

DEASSIGN *[logical-name[:]]*

PARAMETER

logical-name[:]

Specifies the logical name to be deassigned. Logical names can have from 1 to 255 characters. If the logical name contains any characters other than alphanumerics, dollar signs ($), or underscores (_), enclose it in quotation marks (" "). The *logical-name* parameter is required unless you use the /ALL qualifier.

If the *logical-name* parameter ends with a colon (:), the command interpreter ignores the colon. (Note that the ASSIGN and ALLOCATE commands remove a trailing colon, if present, from a logical name before placing the name in a logical name table.) If a colon is present in the logical name, you must type two colons in the *logical-name* parameter of the DEASSIGN command (for example, DEASSIGN FILE::).

To delete a logical name table, specify the table name as the *logical-name* parameter. You must also use the /TABLE qualifier to indicate the logical name directory table where the table name is entered.

DESCRIPTION

The DEASSIGN command cancels a logical name assignment that was made with one of the following commands: ALLOCATE, ASSIGN, DEFINE, or MOUNT. The DEASSIGN command also deletes a logical name table that was created with the CREATE/NAME_TABLE command. You can use the /ALL qualifier with DEASSIGN to cancel all logical names in a specified table. If you use the /ALL qualifier and do not specify a table, then all names in the process table (except names created by the command interpreter) are deassigned; that is, all names entered at the indicated access mode or an outer access mode are deassigned.

To specify the logical name table from which you want to deassign a logical name, use the /PROCESS, the /JOB, the /GROUP, the /SYSTEM, or the /TABLE qualifier. If you enter more than one of these qualifiers, only the last one entered is accepted. If entries exist for the specified logical name in more than one logical name table, the name is deleted from only the last logical name table specified on the command line. If you do not specify a logical name table, the default is the /TABLE=LNM$PROCESS qualifier.

To specify the access mode of the logical name you want to deassign, use the /USER_MODE, the /SUPERVISOR_MODE, or the /EXECUTIVE_ MODE qualifier. If you enter more than one of these qualifiers, only the last one is accepted. If you do not specify a mode, the DEASSIGN command deletes a supervisor-mode name. When you deassign a logical

name, any identical names created with outer access modes in the same logical name table are also deleted.

You must have SYSNAM (system logical name) privilege to deassign an executive-mode logical name. If you specify the /EXECUTIVE_MODE qualifier and you do not have SYSNAM privilege, then the DEASSIGN command ignores the qualifier and attempts to deassign a supervisor-mode logical name.

All process-private logical names and logical name tables are deleted when you log out of the system. User-mode entries within the process logical name table are deassigned when any image exits. The logical names in the job table, and the job table itself, are deleted when you log off the system.

Names in all other shareable logical name tables remain there until they are explicitly deassigned, regardless of whether they are user-, supervisor-, or executive-mode names. You must have write (W) access to a shareable logical name table to delete any name in that table.

If you delete a logical name table, all the logical names in the table are also deleted. Also, any descendant tables are deleted. To delete a shareable logical name table, you must have the user privilege SYSPRV (system privilege) or you must have delete (D) access to the table.

QUALIFIERS

/ALL
Deletes all logical names in the same or an outer (less privileged) access mode. If no logical name table is specified, the default is the process table, LNM$PROCESS. If you specify the /ALL qualifier, you cannot enter a *logical-name* parameter.

/EXECUTIVE_MODE
Requires SYSNAM (system logical name) privilege to deassign executive-mode logical names.

Deletes only entries that were created in the specified mode or an outer (less privileged) mode. If you do not have SYSPRV (system privilege) privilege for executive mode, a supervisor-mode operation is assumed.

/GROUP
Requires GRPNAM (group logical name) or SYSPRV privilege to delete entries from the group logical name table.

Indicates that the specified logical name is in the group logical name table. The /GROUP qualifier is synonymous with the /TABLE=LNM$GROUP qualifier.

/JOB
Indicates that the specified logical name is in the jobwide logical name table. The /JOB qualifier is synonymous with the /TABLE=LNM$JOB qualifier. If you do not explicitly specify a logical name table, the default is the /PROCESS qualifier.

You should not deassign jobwide logical name entries that were made by the system at login time, for example, SYS$LOGIN, SYS$LOGIN_DEVICE, and SYS$SCRATCH. However, if you assign new equivalence names for these logical names (that is, create new logical names in outer access modes), you can deassign the names you explicitly created.

/PROCESS (default)

Indicates that the specified logical name is in the process logical name table. The /PROCESS qualifier is synonymous with the /TABLE=LNM$PROCESS qualifier.

You cannot deassign logical name table entries that were made by the command interpreter, for example, SYS$INPUT, SYS$OUTPUT, and SYS$ERROR. However, if you assign new equivalence names for these logical names (that is, you create new logical names in outer access modes), you can deassign the names you explicitly created.

/SUPERVISOR_MODE (default)

Deletes entries in the specified logical name table that were created in supervisor mode. If you specify the /SUPERVISOR_MODE qualifier, the DEASSIGN command also deassigns user-mode entries with the same name.

/SYSTEM

Requires SYSNAM (system logical name) or SYSPRV (system privilege) privilege to delete entries from the system logical name table.

Indicates that the specified logical name is in the system logical name table. The /SYSTEM qualifier is synonymous with the /TABLE=LNM$SYSTEM qualifier.

/TABLE=name

Requires write (W) access to the table to delete a shareable logical name. Requires SYSPRV privilege or delete (D) access to delete a shareable logical name table.

Specifies the table from which the logical name is to be deleted. Defaults to LNM$PROCESS. The table can be the process, group, job, or system table, one of the directory tables, or the name of a user-created table. (The process, job, group, and system logical name tables should be referred to by the logical names LNM$PROCESS, LNM$JOB, LNM$GROUP, and LNM$SYSTEM, respectively.)

The /TABLE qualifier also can be used to delete a logical name table. To delete a process-private table, enter the following command:

```
$ DEASSIGN/TABLE=LNM$PROCESS_DIRECTORY table-name
```

To delete a shareable table, enter the following command:

```
$ DEASSIGN/TABLE=LNM$SYSTEM_DIRECTORY table-name
```

To delete a shareable logical name table, you must have delete (D) access to the table or write (W) access to the directory table in which the name of the shareable table is cataloged.

If you do not explicitly specify the /TABLE qualifier, the default is the /TABLE=LNM$PROCESS qualifier.

/USER_MODE

Deletes entries in the process logical name table that were created in user mode. If you specify the /USER_MODE qualifier, the DEASSIGN command can deassign only user-mode entries.

EXAMPLES

1 $ DEASSIGN MEMO

> The DEASSIGN command in this example deassigns the process logical name MEMO.

2 $ DEASSIGN/ALL

> The DEASSIGN command in this example deassigns all process logical names that were created in user and supervisor mode. This command does not, however, delete the names that were placed in the process logical name table in executive mode by the command interpreter (for example, SYS$INPUT, SYS$OUTPUT, SYS$ERROR, SYS$DISK, and SYS$COMMAND).

3 $ DEASSIGN/TABLE=LNM$PROCESS_DIRECTORY TAX

> The DEASSIGN command in this example deletes the logical name table TAX, and any descendant tables. When you delete a logical name table, you must specify either the /TABLE=LNM$PROCESS_DIRECTORY or the /TABLE=LNM$SYSTEM_DIRECTORY qualifier, because the names of all tables are contained in these directories.

4 $ ASSIGN USER_DISK: COPY
    $ DEASSIGN COPY

> The ASSIGN command in this example equates the logical name COPY with the device USER_DISK and places the names in the process logical name table. The DEASSIGN command deletes the logical name.

5 $ DEFINE SWITCH: TEMP
    $ DEASSIGN SWITCH::

> The DEFINE command in this example places the logical name SWITCH: in the process logical name table. The trailing colon is retained as part of the logical name. Two colons are required on the DEASSIGN command to delete this logical name because the DEASSIGN command removes one trailing colon, and the other colon is needed to match the characters in the logical name.

6 $ ASSIGN/TABLE=LNM$GROUP DBA1: GROUP_DISK
    $ DEASSIGN/PROCESS/GROUP GROUP_DISK

> The ASSIGN command in this example places the logical name GROUP_DISK in the group logical name table. The DEASSIGN command specifies conflicting qualifiers; because the /GROUP qualifier is last, the name is successfully deassigned.

7    $ ASSIGN DALLAS::USER_DISK: DATA
         .
         .
         .
    $ DEASSIGN DATA

The ASSIGN command in this example associates the logical name DATA with the device specification USER_DISK on remote node DALLAS. Subsequent references to the logical name DATA result in references to the disk on the remote node. The DEASSIGN command cancels the logical name assignment.

DEASSIGN/QUEUE

Deassigns a logical queue from a printer or terminal queue and stops the logical queue.

Requires OPER (operator) privilege or execute (E) access to the queue. Cannot be used with batch queues.

FORMAT   **DEASSIGN/QUEUE** *logical-queue-name[:]*

PARAMETER   *logical-queue-name[:]*
Specifies the name of the logical queue that you want to deassign from a specific printer or terminal queue.

DESCRIPTION   Once you enter the DEASSIGN/QUEUE command, the jobs in the logical queue remain pending until the queue is reassigned to another printer queue or device with the ASSIGN/QUEUE command.

EXAMPLE

```
$  ASSIGN/QUEUE LPA0  ASTER
   .
   .
   .
$  DEASSIGN/QUEUE  ASTER
$  ASSIGN/MERGE LPB0  ASTER
```

The ASSIGN/QUEUE command in this example associates the logical queue ASTER with the print queue LPA0. Later, you deassign the logical queue with the DEASSIGN/QUEUE command. The ASSIGN/MERGE command reassigns the jobs from ASTER to the print queue LPB0.

DEBUG

Invokes the VMS Debugger after program execution is interrupted by Ctrl/Y, but only if the /NOTRACEBACK qualifier was not specified with the LINK command when the program was linked. For a complete description of the VMS Debugger, see the *VMS Debugger Manual*.

FORMAT   DEBUG

DECK

Marks the beginning of an input stream for a command or program. The DECK command is required in command procedures when the first nonblank character in any data record in the stream is a dollar sign ($).

Can be used only after a request to execute a command or program that requires input data.

FORMAT   **DECK**

DESCRIPTION   The DECK command marks the data that follows it as input for a command or program. This command is required in command procedures when the first nonblank character in any data record in the input stream is a dollar sign.

The DECK command must be preceded by a dollar sign; the dollar sign must be in the first character position (column 1) of the input record.

The DECK command defines an end-of-file (EOF) indicator only for a single data stream. Using the DECK command enables you to place data records beginning with dollar signs in the input stream. You can place one or more sets of data in the input stream following a DECK command, if each is terminated by an EOF indicator.

After an EOF indicator specified with the /DOLLARS qualifier is encountered, the EOF indicator is reset to the default, that is, to any record beginning with a dollar sign. The default is also reset if an actual EOF occurs for the current command level.

QUALIFIER   **/DOLLARS[=string]**
Sets the EOF indicator to the specified string of 1 to 15 characters. Specify a string if the input data contains one or more records beginning with the string $EOD. Enclose the string in quotation marks (" ") if it contains literal lowercase letters, multiple blanks, or tabs. If you do not specify /DOLLARS, or if you specify /DOLLARS without specifying a string, you must use the EOD command to signal the end-of-file (EOF).

EXAMPLES

1

Input Stream for Program A

$ EOJ
$ PRINT SUMMARY.DAT
$ EOD
$ 99.50
$ 86.42
$ DECK
$ RUN A
$ LINK A
$ FORTRAN A
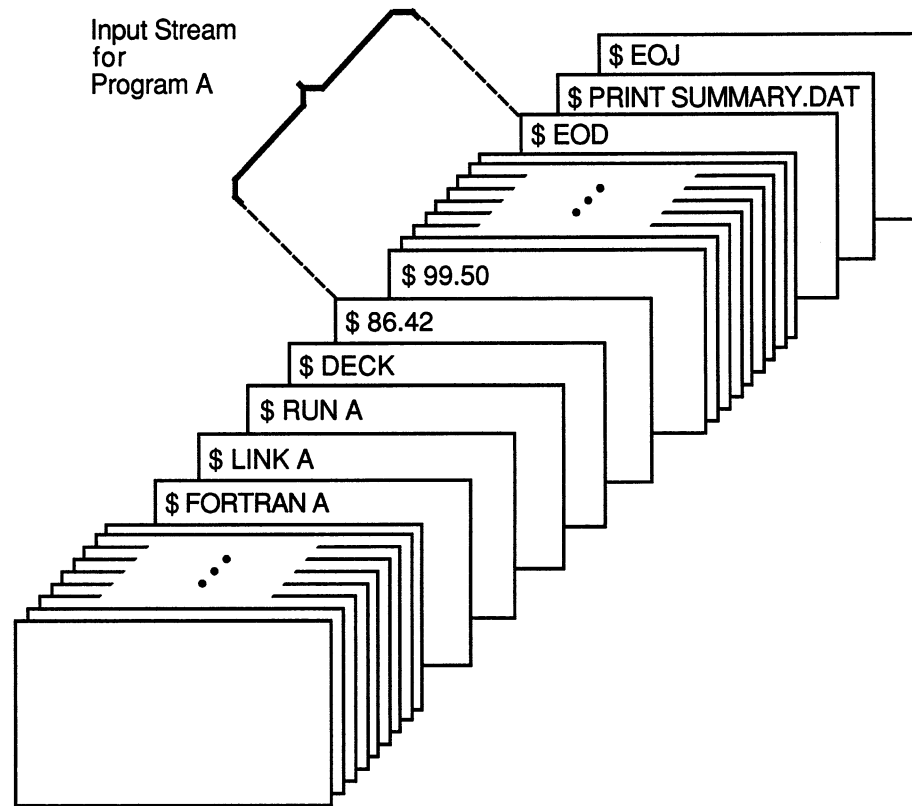
ZK–0783–GE

In this example, the FORTRAN and LINK commands compile and link program A. When the program is run, any data the program reads from the logical device SYS$INPUT is read from the command stream. The DECK command indicates that the input stream can contain dollar signs in column 1 of the record. The EOD command signals end-of-file (EOF) for the data.

2

```
                                              $ EOJ
                                                • •
                                                 •
                                      $ @TEST
                                    %
                              $ PRINT RUNTEST.OUT
                         $ EOD
                            • •
                             •
              $ 99.50
           $ DECK
        $ RUN READFILE
     $ ASSIGN RUNTEST.OUT
     OUTFILE
  $ ASSIGN SYS$INPUT
  INFILE
$ DECK/DOLLARS="%"
$ CREATE TEST.COM
   • •
    •
$ JOB HIGGINS
```
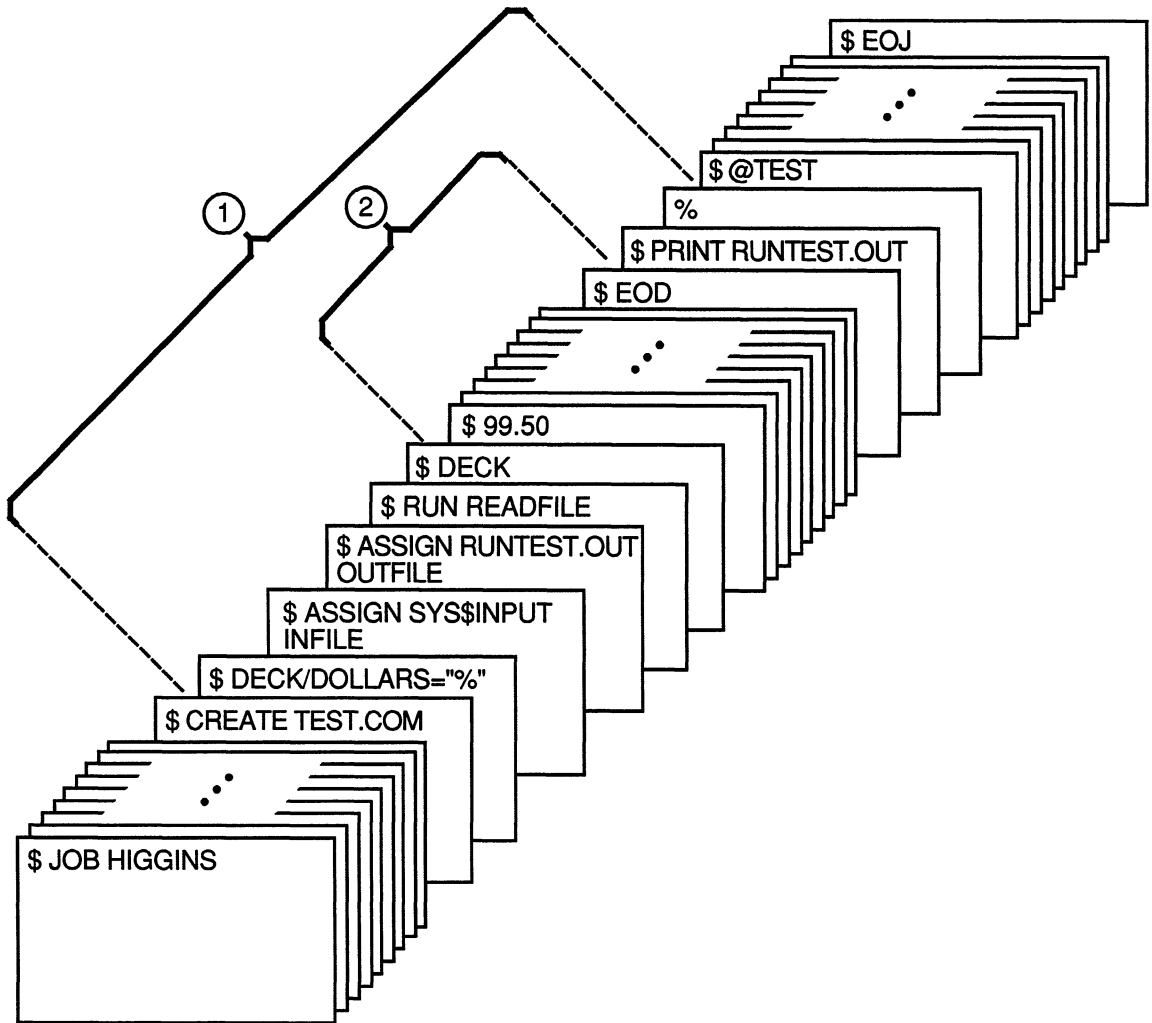
(1) Input stream for CREATE command.

(2) Input stream for program READFILE.

ZK–0784–GE

The CREATE command in this example creates the command procedure
file TEST.COM from lines entered into the input stream. The DECK
/DOLLARS command indicates that the percent sign (%) is the EOF
indicator for the CREATE command. This allows the string $EOD to
be read as an input record, signaling the end of the input for the RUN
command.

DEFINE

Associates equivalence names with a logical name. If you specify an existing logical name, the new equivalence names replace the existing equivalence name.

FORMAT   **DEFINE**   *logical-name equivalence-name[,...]*

PARAMETERS   *logical-name*
Specifies the logical name string, which is a character string containing from 1 to 255 characters. If the logical name is to be entered into the process or system directory logical name tables (LNM$PROCESS_DIRECTORY, LNM$SYSTEM_DIRECTORY), then the name can only have from 1 to 31 alphanumeric characters (including the dollar sign [$] and underscore [_]).

If you specify a colon (:) at the end of a logical name, the DEFINE command saves the colon as part of the logical name. (This is in contrast to the ASSIGN command, which removes the colon before placing the name in a logical name table.) By default, the logical name is placed in the process logical name table.

If the string contains any characters other than uppercase alphanumerics, the dollar sign, or the underscore character, enclose the string in quotation marks (" "). Use two consecutive quotation marks ("") to denote an actual quotation mark. Note that if you enclose a name in quotation marks, the case of alphabetic characters is preserved.

equivalence-name[,...]
Specifies a character string containing from 1 to 255 characters. If the string contains any characters other than uppercase alphanumerics, the dollar sign, or the underscore character, enclose the string in quotation marks. Use two consecutive quotation marks to denote an actual quotation mark. Specifying more than one equivalence name for a logical name creates a search list.

When you specify an equivalence name that will be used as a file specification, you must include the punctuation marks (colons, brackets, periods) that would be required if the equivalence name were used directly as a file specification. Therefore, if you specify a device name as an equivalence name, you must terminate the equivalence name with a colon.

The DEFINE command allows you to assign the same logical name to more than one equivalence name. For example, you can use the same logical name to access different directories on different disks, or to access different files in different directories. When you specify more than one equivalence name for a logical name, you create a search list. See the *VMS DCL Concepts Manual* for more information on search lists.

DESCRIPTION

The DEFINE command creates an entry in a logical name table by defining a logical name to stand for one or more equivalence names. An equivalence name can be a device name, another logical name, a file specification, or any other string.

To specify the logical name table where you want to enter a logical name, use the /PROCESS, the /GROUP, the /SYSTEM, the /JOB, or the /TABLE qualifier. If you enter more than one of these qualifiers, only the last one entered is accepted. If you do not specify a table, the default is the /TABLE=LNM$PROCESS qualifier.

To specify the access mode of the logical name you are creating, use the /USER_MODE, the /SUPERVISOR_MODE, or the /EXECUTIVE_MODE qualifier. If you enter more than one of these qualifiers, only the last one entered is accepted. If you do not specify an access mode, a supervisor-mode name is created. You can create a logical name in the same mode as the table in which you are placing the name, or in an outer mode. (User mode is the outermost mode; executive mode is the innermost mode.)

You can enter more than one logical name with the same name in the same table, as long as each name has a different access mode. (However, if an existing logical name within a table has the NO_ALIAS attribute, you cannot use the same name to create a logical name in an outer mode in this table.)

If you create a logical name with the same name, in the same table, and in the same mode as an existing name, the new logical name assignment replaces the existing assignment.

You can also use the ASSIGN command to create logical names. To delete a logical name from a table, use the DEASSIGN command.

Note: **Avoid assigning a logical name that matches the file name of an executable image in SYS$SYSTEM:. Such an assignment prohibits you from invoking that image.**

For additional information on how to create and use logical names, see the *VMS DCL Concepts Manual*.

QUALIFIERS

/EXECUTIVE_MODE
Requires SYSNAM (system logical name) privilege to create an executive-mode logical name.

Creates an executive-mode logical name in the specified table.

If you specify the /EXECUTIVE_MODE qualifier and you do not have SYSNAM privilege, the DEFINE command ignores the qualifier and creates a supervisor-mode logical name. The mode of the logical name must be the same or less privileged than the mode of the table in which you are placing the name.

/GROUP

Requires GRPNAM (group logical name) or SYSPRV (system privilege) privilege to place a name in the group logical name table.

Places the logical name in the group logical name table. Other users who have the same group number in their user identification codes (UICs) can access the logical name. The /GROUP qualifier is synonymous with the /TABLE=LNM$GROUP qualifier.

/JOB

Places the logical name in the jobwide logical name table. All processes in the same job tree as the process that created the logical name can access the logical name. The /JOB qualifier is synonymous with the /TABLE=LNM$JOB qualifier.

/LOG (default)
/NOLOG

Displays a message when a new logical name supersedes an existing name.

/NAME_ATTRIBUTES[=(keyword[,...])]

Specifies attributes for a logical name. By default, no attributes are set. Possible keywords are as follows:

CONFINE	The logical name is not copied into a spawned subprocess. This qualifier is relevant only for logical names in a private table.
	The logical name inherits the CONFINE attribute from the logical name table where it is entered; if the logical name table is "confined," then all names in the table are "confined."
NO_ALIAS	A logical name cannot be duplicated in the specified table in a less privileged access mode; any previously created identical names in an outer (less privileged) access mode within the specified table are deleted.

If you specify only one keyword, you can omit the parentheses. Only the attributes you specify are set.

/PROCESS (default)

Places the logical name in the process logical name table. The /PROCESS qualifier is synonymous with the /TABLE=LNM$PROCESS qualifier.

/SUPERVISOR_MODE (default)

Creates a supervisor-mode logical name in the specified table. The mode of the logical name must be the same as or less privileged than the mode of the table in which you are placing the name.

/SYSTEM

Requires SYSNAM (system logical name) or SYSPRV (system privilege) privilege to place a name in the system logical name table.

Places the logical name in the system logical name table. All system users can access the logical name. The /SYSTEM qualifier is synonymous with the /TABLE=LNM$SYSTEM qualifier.

/TABLE=name
Requires write (W) access to the table to specify the name of a shareable logical name table.

Specifies the name of the logical name table in which the logical name is to be entered. You can use the /TABLE qualifier to specify a user-defined logical name table (created with the CREATE/NAME_TABLE command); to specify the process, job, group, or system logical name tables; or to specify the process or system logical name directory tables.

If you specify the table name using a logical name that has more than one translation, the logical name is placed in the first table found. For example, if you specify DEFINE/TABLE=LNM$FILE_DEV and LNM$FILE_DEV is equated to LNM$PROCESS, LNM$JOB, LNM$GROUP, and LNM$SYSTEM, then the logical name is placed in LNM$PROCESS.

The default is the /TABLE=LNM$PROCESS qualifier.

/TRANSLATION_ATTRIBUTES[=(keyword[,...])]
Equivalence-name qualifier.

Specifies one or more attributes that modify an equivalence string of the logical name. Possible keywords are as follows:

CONCEALED   Indicates that the equivalence string is the name of a concealed device. When a concealed device name is defined, the system displays the logical name, rather than the equivalence string, in messages that refer to the device.

TERMINAL   Logical name translation should terminate with the current equivalence string; indicates that the equivalence string should not be translated iteratively.

If you specify only one keyword, you can omit the parentheses. Only the attributes you specify are set.

Note that different equivalence strings of a logical name can have different translation attributes.

/USER_MODE
Creates a user-mode logical name in the specified table.

User-mode logical names created within the process logical name tables are used for the execution of a single image; for example, you can create a user-mode logical name to allow an image executing in a command procedure to redefine SYS$INPUT. User-mode entries are deleted from the process logical name table when any image executing in the process exits (that is, after a DCL command or user program that executes an image completes execution).

EXAMPLES

1   `$ DEFINE/USER_MODE TM1 $DISK1:[ACCOUNTS.MEMOS]WATER.TXT`

In this example, the DEFINE command defines TM1 as equivalent to a file specification. After the next image runs, the logical name TM1 is automatically deassigned.

2   `$ DEFINE MEMO $DISK1:[ACCOUNTS.MEMO]`

> In this example, the DEFINE command defines the logical name MEMO as equivalent to the partial file specification $DISK1:[ACCOUNTS.MEMO].

3   `$ DEFINE PROCESS_NAME  LIBRA`
          `$ RUN WAKE`

> In this example, the DEFINE command places the logical name PROCESS_NAME in the process logical name table with an equivalence name of LIBRA. The logical name is created in supervisor mode. The program WAKE translates the logical name PROCESS_NAME to perform some special action on the process named LIBRA.

4   `$ DEFINE TEMP: XXX1:`
           .
           .
           .
      `$ DEASSIGN TEMP::`

> In this example, the DEFINE command creates an equivalence name for the logical name TEMP: and places the name in the process logical name table. The colon is retained as part of the logical name. The DEASSIGN command deletes the logical name. Note that two colons are required on the logical name in the DEASSIGN command. One colon is deleted by the DEASSIGN command. The other colon is kept as part of the logical name.

5   `$ DEFINE PORTLAND PRTLND::YYY0:[DECNET.DEMO.COM]`

> In this example, the DEFINE command places the logical name PORTLAND in the process logical name table with an equivalence name of PRTLND::YYY0:[DECNET.DEMO.COM]. Subsequent references to the logical name PORTLAND result in the correspondence between the logical name PORTLAND and the node, disk, and subdirectory specified.

6   `$ DEFINE LOCAL "BOSTON""JOHN_SMITH JKS""::"`

> In this example, the DEFINE command places the logical name LOCAL in the process logical name table with a remote node equivalence name of BOSTON"JOHN_SMITH JKS"::. To satisfy conventions for local DCL command string processing, you must use three sets of quotation marks. The quotation marks ensure that access control information is enclosed in one set of quotation marks in the equivalence name.

7   `$ DEFINE MYDISK XXX0:[MYDIR], YYY0:[TESTDIR]`

> In this example, the DEFINE command places the logical name MYDISK in the process logical name table with two equivalence names: XXX0:[MYDIR] and YYY0:[TESTDIR].

```
8   $   CREATE/NAME_TABLE TABLE1
    $   DEFINE/TABLE=LNM$PROCESS_DIRECTORY LNM$FILE_DEV -
   _$   TABLE1,LNM$PROCESS,LNM$JOB,LNM$GROUP,LNM$SYSTEM
    $   DEFINE/TABLE=TABLE1 -
   _$   /TRANSLATION_ATTRIBUTES=CONCEALED   WORK_DISK DBA1:
```

In this example, the CREATE/NAME_TABLE command creates the process private logical name table TABLE1.

The first DEFINE command ensures that TABLE1 is searched first in any logical name translation of a device or file specification (because TABLE1 is the first item in the equivalence string for the logical name LNM$FILE_DEV, which determines the default search sequence of logical name tables whenever a device or file specification is translated).

The second DEFINE command assigns the logical name WORK_DISK to the physical device DBA1 and places the name in TABLE1. The logical name has the concealed attribute. Therefore, the logical name WORK_DISK is displayed in system messages.

```
9   $   CREATE/NAME_TABLE SPECIAL
    $   DEFINE/TABLE=LNM$PROCESS_DIRECTORY LNM$FILE_DEV -
   _$    SPECIAL,LNM$PROCESS,LNM$JOB,LNM$GROUP,LNM$SYSTEM
    $   DEFINE/TABLE=LNM$PROCESS_DIRECTORY TAB SPECIAL
    $   DEFINE/TABLE=TAB REPORT [CHELSEA]STORES
    $   SHOW LOGICAL/TABLE=SPECIAL REPORT
       "REPORT" = "[CHELSEA]STORES" (SPECIAL)
```

In this example, the CREATE/NAME_TABLE command is used to create a new logical name table called SPECIAL. This table is defined in the process directory, LNM$PROCESS_DIRECTORY.

The first DEFINE command ensures that SPECIAL is searched first in any logical name translation of a device or file specification (because SPECIAL is the first item in the equivalence string for the logical name LNM$FILE_DEV, which determines the default search sequence of logical name tables whenever a device or file specification is translated). The logical name LNM$FILE_DEV is placed in the process directory, LNM$PROCESS_DIRECTORY.

With the next DEFINE command, a new logical name, TAB, is defined. TAB translates to the string SPECIAL, which identifies a logical name table. You must define TAB in the process directory because it translates iteratively to a logical name table.

Next, the logical name REPORT is placed into the logical name table TAB. Because TAB translates to the table SPECIAL, the name REPORT is entered into SPECIAL table. The SHOW LOGICAL command verifies that the name REPORT has been entered into the table SPECIAL.

Note that you can redefine TAB so it translates to a different table. Therefore, if you run different programs that use the name TAB as a table name, you can change the actual tables where the names are entered or referenced.

DEFINE/CHARACTERISTIC

Assigns a numeric value to a queue characteristic. The /CHARACTERISTIC qualifier is required. If a value has been assigned to the characteristic, the DEFINE/CHARACTERISTIC command alters the assignment of the existing characteristic.

Requires OPER (operator) privilege.

FORMAT   **DEFINE/CHARACTERISTIC**   *characteristic-name*
characteristic-number

PARAMETERS   *characteristic-name*
Assigns a name to the characteristic being defined. The characteristic name can be the name of an existing characteristic or a string of 1 to 31 characters that defines a new characteristic. The character string can include any uppercase and lowercase letters, digits, the dollar sign ($), and the underscore (_), and must include at least one alphabetic character.

characteristic-number
Assigns a number in the range 0 to 127 to the characteristic being defined.

DESCRIPTION   The system manager or operator uses the DEFINE/CHARACTERISTIC command to assign a name and number to a particular characteristic for queues in the system. Characteristics can refer to any attribute of a print or batch job that is meaningful for your environment. The name and number of a characteristic are arbitrary, but they must be unique for that characteristic.

After characteristics have been defined, they can be associated with print or batch jobs and execution queues. For information on specifying characteristics with jobs, see the description of the /CHARACTERISTICS qualifier of the PRINT and SUBMIT commands.

To find out what characteristics are currently defined for the system, use the SHOW QUEUE/CHARACTERISTICS command. To find out which characteristics have been specified for a particular queue, use the SHOW QUEUE/FULL command. For information on associating characteristics with queues, see the descriptions of the /CHARACTERISTICS qualifier of the INITIALIZE/QUEUE, SET QUEUE, and START/QUEUE commands.

The DELETE/CHARACTERISTIC command deletes a previously defined characteristic.

For more information on specifying queue characteristics, see the *Guide to Maintaining a VMS System*.

EXAMPLE

```
$  DEFINE/CHARACTERISTIC REDINK 3
```

The DEFINE/CHARACTERISTIC command in this example defines the characteristic REDINK with the number 3. When a user enters the command PRINT/CHARACTERISTICS=REDINK (or PRINT /CHARACTERISTICS=3), the job is printed only if the printer queue has been established with the REDINK or 3 characteristic.

DEFINE/FORM

Assigns a numeric value and attributes to a print form name. The /FORM qualifier is required. If a value has been assigned already to the form name, the DEFINE/FORM command alters the definition of the existing form.

Requires OPER (operator) privilege.

FORMAT

DEFINE/FORM *form-name form-number*

PARAMETERS

form-name

Assigns a name to the form being defined. The form name can be the name of an existing form type or a string of 1 to 31 characters that defines a new form type. The character string can include any uppercase and lowercase letters, digits, the dollar sign ($), and the underscore (_), and must include at least one alphabetic character.

form-number

Assigns a number in the range 0 to 2,147,483,647 to the form being defined. The DEFAULT form, which is defined automatically when the system is bootstrapped, is assigned number zero.

DESCRIPTION

The system manager or operator uses the DEFINE/FORM command to assign a name and number to a type of paper stock and printing area for use with printer or terminal queues. When a new queue file is created, the system defines a form named DEFAULT with a form number of zero and all the default attributes.

Some DEFINE/FORM qualifiers specify the area for printing. The LEFT and RIGHT options of the /MARGIN qualifier and the /WIDTH qualifier determine the number of characters per line. Using the RIGHT option of the MARGIN qualifier and the /WIDTH qualifier, you can affect the point at which lines of text wrap. (You cannot use the LEFT and RIGHT options of the /MARGIN qualifier and the /WIDTH qualifier for filling or formatting the text, however.)

You also can use the DEFINE/FORM command to specify different types of paper stock. The /DESCRIPTION qualifier enables you to describe more fully the form name.

After forms have been defined, they can be associated with print jobs and output execution queues. For information on specifying forms with jobs, see the description of the PRINT/FORM command.

To find out what forms have been defined for the system, use the SHOW QUEUE/FORM command. To find out which form is mounted currently on a particular queue and which form is specified as that queue's default form, use the SHOW QUEUE/FULL command. For information on associating forms with queues, see the descriptions of the /DEFAULT

DEFINE/FORM

and /FORM_MOUNTED qualifiers of the INITIALIZE/QUEUE, SET QUEUE, and START/QUEUE commands.

For more information on how to use forms to control print jobs, see the *Guide to Maintaining a VMS System*.

QUALIFIERS	

/DESCRIPTION=string

A string of up to 255 characters used to provide operator information about the form. The default string is the specified form name.

The string can be used to define the form type more specifically. For example, if you have form names such as LETTER1, LETTER2, and LETTER3, the /DESCRIPTION qualifier could be used to let the users and operators know that LETTER1 refers to the standard corporate letterhead paper (8.5 inches x 11 inches), LETTER2 refers to the smaller corporate letterhead paper (6 inches x 9 inches), and LETTER3 refers to the president's personalized letterhead paper.

Enclose strings containing lowercase letters, blanks, or other nonalphanumeric characters (including spaces) in quotation marks (" ").

/LENGTH=n

Specifies the physical length of a form page in lines. The default page length is 66 lines, which assumes a standard page length of 11 inches with 6 lines of print per inch. The parameter n must be a positive integer greater than zero and not more than 255.

The print symbiont sets the page length of the device equal to the form length. This enables the driver to compute the number of line feeds for devices lacking mechanical form feed.

/MARGIN=(option[,...])

Specifies one or more of the four margin options: BOTTOM, LEFT, RIGHT, and TOP.

BOTTOM=n   Specifies the number of blank lines between the end of the print image area and the end of the physical page; the value of n must be between 0 and the value of the /LENGTH qualifier. The default value is 6, which generally means a 1-inch bottom margin.

LEFT=n   Specifies the number of blank columns between the leftmost printing position and the print image area; the value of n must be between 0 and the value of the /WIDTH qualifier. The default is 0, which means that the print image area starts as far to the left of the paper as the printer can go.

RIGHT=n   Specifies the number of blank columns between the /WIDTH qualifier and the image area; the value of n must be between 0 and the value of the /WIDTH qualifier. When determining the value of the RIGHT option, start at the /WIDTH value and count to the left. The default value is 0, which means that the print image extends as far to the right as the /WIDTH value.

TOP=n   Specifies the number of blank lines between the top of the physical page and the top of the print image; the value of n must be between 0 and the value of the /LENGTH qualifier. The default value is 0, which generally means that there is no top margin.

/PAGE_SETUP=(module[,...])
/NOPAGE_SETUP (default)

Specifies one or more modules that set up the device at the start of each page. The modules are located in the device control library. While the form is mounted, the system extracts the specified module and copies it to the printer before each page is printed.

/SETUP=(module[,...])

Specifies one or more modules that set up the device at the start of each file. The modules are located in the device control library. While the form is mounted, the system extracts the specified module and copies it to the printer before each file is printed.

/SHEET_FEED
/NOSHEET_FEED (default)

Specifies that print jobs pause at the end of every physical page so that a new sheet of paper can be inserted.

/STOCK=string

Specifies the type of paper stock to be associated with the form. The *string* parameter can be a string of 1 to 31 characters, including the dollar sign, underscore, and all alphanumeric characters. If you specify the /STOCK qualifier you must specify the name of the stock to be associated with the form. If you do not specify the /STOCK qualifier, the name of the stock will be the same as the name of the form.

You can create any string that you want. However, when you are creating forms with the same stock, be sure that the /STOCK string is identical in all the DEFINE/FORM commands that refer to the same type of paper.

If you are defining a number of forms to provide different formatting options, specify the same stock type for each form. If you specify the same stock type for each form, jobs that request any of these forms will print on the same queue.

/TRUNCATE (default)
/NOTRUNCATE

Discards any characters that exceed the current line length (specified by the /WIDTH and /MARGIN=RIGHT qualifiers). The /TRUNCATE qualifier is incompatible with the /WRAP qualifier. If you specify both the /NOTRUNCATE and /NOWRAP qualifiers, the printer prints as many characters on a line as possible. This combination of qualifiers is useful for some types of graphics output.

/WIDTH=n

Specifies the physical width of the paper in terms of columns or character positions. The parameter n must be an integer from 0 to 65,535; the default value is 132.

Any lines exceeding this value wrap if the /WRAP qualifier is in effect or are truncated if the /TRUNCATE qualifier is in effect. (If both the /NOTRUNCATE and /NOWRAP qualifiers are in effect, lines print as far as possible.)

The /MARGIN=RIGHT qualifier overrides the /WIDTH qualifier when determining when to wrap lines of text.

/WRAP
/NOWRAP (default)

Causes lines that exceed the current line length (specified by the /WIDTH and /MARGIN=RIGHT qualifiers) to wrap onto the next line. The /WRAP qualifier is incompatible with the /TRUNCATE qualifier. If you specify both the /NOWRAP and /NOTRUNCATE qualifiers, the printer prints as many characters on a line as possible. This combination of qualifiers is useful for some types of graphics output.

EXAMPLE

```
$  DEFINE/FORM /MARGIN=(TOP=6,LEFT=10) CENTER 3
```

The DEFINE/FORM command in this example defines the form CENTER to have a top margin of 6 and a left margin of 10. The defaults remain in effect for both bottom margin (6) and right margin (0). The form is assigned the number 3.

DEFINE/KEY

Associates an equivalence string and a set of attributes with a key on the terminal keyboard. The /KEY qualifier is required.

FORMAT   **DEFINE/KEY** *key-name equivalence-string*

PARAMETERS   *key-name*

Specifies the name of the key that you are defining. All definable keys on VT52 terminals are located on the numeric keypad. On VT100-series terminals, you can define the left and right arrow keys as well as all the keys on the numeric keypad. On terminals with LK201 keyboards, the following three types of keys can be defined:

- Keys on the numeric keypad

- Keys on the editing keypad (except the up and down arrow keys)

- Keys on the function key row across the top of the keyboard (except keys F1 to F5)

The following table lists the key names in column one. The remaining three columns indicate the key designations on the keyboards of the three different types of terminals that allow key definitions.

Key Name	LK201	VT100-Series	VT52
PF1	PF1	PF1	[blue]
PF2	PF2	PF2	[red]
PF3	PF3	PF3	[gray]
PF4	PF4	PF4	- -
KP0, KP1, ..., KP9	0, 1, ..., 9	0, 1, ..., 9	0, 1, ..., 9
Period	.	.	.
Comma	,	,	n/a
Minus	-	-	n/a
Enter	Enter	ENTER	ENTER
Left	←	←	←
Right	→	→	→
Find (E1)	Find	—	—
Insert Here (E2)	Insert Here	—	—
Remove (E3)	Remove	—	—
Select (E4)	Select	—	—
Prev Screen (E5)	Prev Screen	—	—

Key Name	LK201	VT100-Series	VT52
Next Screen (E6)	Next Screen	—	—
Help	Help	—	—
Do	Do	—	—
F6, F7, ..., F20	F6, F7, ..., F20	—	—

Some definable keys are enabled for definition all the time. Others, including KP0 to KP9, Period, Comma, and Minus, must be enabled for definition purposes. You must enter either the SET TERMINAL /APPLICATION or the SET TERMINAL/NONUMERIC command before using these keys.

On LK201 keyboards, you cannot define the up and down arrow keys or function keys F1 to F5. The left and right arrow keys and the F6 to F14 keys are reserved for command line editing. You must enter the SET TERMINAL/NOLINE_EDITING command before defining these keys. You can also press Ctrl/V to enable keys F7 to F14. Note that Ctrl/V will not enable the F6 key.

equivalence-string
Specifies the character string to be processed when you press the key. Enclose the string in quotation marks (" ") to preserve spaces and lowercase characters.

DESCRIPTION

The DEFINE/KEY command enables you to assign definitions to the peripheral keys on certain terminals. The terminals include VT52s, the VT100 series, and terminals with LK201 keyboards.

To define keys on the numeric keypads of these terminals, you must first enter the SET TERMINAL/APPLICATION or SET TERMINAL /NONUMERIC command. When your terminal has this setting, the system interprets the keystrokes from keypad keys differently. For example, with SET TERMINAL/NONUMERIC in effect, pressing the 1 key on the keypad does not send the character "1" to the system.

The equivalence string definition can contain different types of information. Definitions often consist of DCL commands. For example, you can assign SHOW TIME to the zero key. When you press 0, the system displays the current date and time. Other definitions can consist of text strings to be appended to command lines. When you define a key to insert a text string, use the /NOTERMINATE qualifier so that you can continue typing more data after the string has been inserted.

In most instances you will want to use the echo feature. The default setting is /ECHO. With /ECHO set, the key definition is displayed on the screen each time you press the key.

You can use the /STATE qualifier to increase the number of key definitions available on your terminal. The same key can be assigned any number of definitions, as long as each definition is associated with a different state. State names can contain any alphanumeric characters, dollar signs, and underscores. Be sure to create a state name that is easy to remember and type and, if possible, one that might remind you of the types of definitions

you created for that state. For example, you can create a state called SETSHOW. The key definitions for this state might all refer to various DCL SET and SHOW commands. If you are used to the EDT Editor, you might define a state as GOLD. Then, using the /IF_STATE qualifier, you can assign different definitions to keys used in combination with a key defined as GOLD.

The SET KEY command changes the keypad state. Use the SHOW KEY command to display key definitions and states.

QUALIFIERS	

/ECHO (default)
/NOECHO
Displays the equivalence string on your screen after the key has been pressed. You cannot use the /NOECHO qualifier with the /NOTERMINATE qualifier.

/ERASE
/NOERASE (default)
Determines whether the current line is erased before the key translation is inserted.

/IF_STATE=(state-name,...)
/NOIF_STATE
Specifies a list of one or more states, one of which must be in effect for the key definition to work. The /NOIF_STATE qualifier has the same meaning as /IF_STATE=current_state. The state name is an alphanumeric string. States are established with the /SET_STATE qualifier or the SET KEY command. If you specify only one state name, you can omit the parentheses. By including several state names, you can define a key to have the same function in all the specified states.

/LOCK_STATE
/NOLOCK_STATE (default)
Specifies that the state set by the /SET_STATE qualifier remain in effect until explicitly changed. (By default, the /SET_STATE qualifier is in effect only for the next definable key you press or the next read-terminating character that you type.) This qualifier can be specified only with the /SET_STATE qualifier.

/LOG (default)
/NOLOG
Displays a message indicating that the key definition has been successfully created.

/SET_STATE=state-name
/NOSET_STATE (default)
Causes the specified state-name to be set when the key is pressed. (By default, the current locked state is reset when the key is pressed.) If you have not included this qualifier with a key definition, you can use the SET KEY command to change the current state. The state name can be any alphanumeric string; specify the state as a character string enclosed in quotation marks.

/TERMINATE
/NOTERMINATE (default)

Specifies whether the current equivalence string is to be processed immediately when the key is pressed (equivalent to entering the string and pressing the Return key). By default, you can press other keys before the definition is processed. This allows you to create key definitions that insert text into command lines, after prompts, or into other text that you are entering.

EXAMPLES

1
```
$  DEFINE/KEY PF3 "SHOW TIME" /TERMINATE
%DCL-I-DEFKEY, DEFAULT key PF3 has been defined
$  PF3
$ SHOW TIME
   19-APR-1990 14:43:59
```

In this example, the DEFINE/KEY command defines the PF3 key on the keypad to perform the SHOW TIME command. DEFAULT refers to the default state.

2
```
$  DEFINE/KEY PF1 "SHOW " /SET_STATE=GOLD/NOTERMINATE/ECHO
%DCL-I-DEFKEY, DEFAULT key PF1 has been defined
$  DEFINE/KEY PF1 " DEFAULT" /TERMINATE/IF_STATE=GOLD/ECHO
%DCL-I-DEFKEY, GOLD key PF1 has been defined
$  PF1
$  PF1
$ SHOW DEFAULT
DISK1:[JOHN.TEST]
```

In this example, the first DEFINE/KEY command defines the PF1 key to be the string SHOW. The state is set to GOLD for the subsequent key. The /NOTERMINATE qualifier instructs the system not to process the string when the key is pressed. The second DEFINE/KEY command defines the use of the PF1 key when the keypad is in the GOLD state. When the keypad is in the GOLD state, pressing PF1 causes the current read to be terminated.

If you press the PF1 key twice, the system displays and processes the SHOW DEFAULT command.

The word DEFAULT in the second line of the example indicates that the PF1 key has been defined in the default state. Note the space before the word DEFAULT in the second DEFINE/KEY command. If the space is omitted, the system fails to recognize DEFAULT as the keyword for the SHOW command.

3
```
$  SET KEY/STATE=ONE
%DCL-I-SETKEY, keypad state has been set to ONE
$  DEFINE/KEY PF1 "ONE"
%DCL-I-DEFKEY, ONE key PF1 has been defined
$  DEFINE/KEY/IF_STATE=ONE PF1 "ONE"
%DCL-I-DEFKEY, ONE key PF1 has been defined
```

This example shows two ways to define the PF1 key to be "ONE" for state ONE.

The second DEFINE/KEY command shows the preferred method for defining keys. This method eliminates the possibility of error by specifying the state in the same command as the key definition.

DELETE

Deletes one or more files from a mass storage disk volume.

FORMAT   **DELETE**   *filespec[,...]*

PARAMETER   ***filespec[,...]***
Specifies the names of one or more files to be deleted from a mass storage
disk volume. The first file specification must contain an explicit or
default directory specification plus an explicit file name, file type, and
version number. Subsequent file specifications need contain only a version
number; the defaults will come from the preceding specification. Wildcard
characters can be used in any of the file specification fields.

If you omit the directory specification or device name, the current default
device and directory are assumed.

If the file specification contains a null version number (a semicolon [;]
followed by no file version number), a version number of 0, or one or more
spaces in the version number, the latest version of the file is deleted.

To delete more than one file, separate the file specifications with either
commas (,) or plus signs (+).

QUALIFIERS   ***/BACKUP***
Modifies the time value specified with the /BEFORE or the /SINCE
qualifier. The /BACKUP qualifier selects files according to the dates
of their most recent backups. This qualifier is incompatible with the
/CREATED, /EXPIRED, and /MODIFIED qualifiers, which also allow you
to select files according to time attributes. If you specify none of these four
time qualifiers, the default is the /CREATED qualifier.

/BEFORE[=time]
Selects only those files dated prior to the specified time. You can specify
time as an absolute time, as a combination of absolute and delta times,
or as one of the following keywords: TODAY (default), TOMORROW, or
YESTERDAY. Specify one of the following qualifiers with the /BEFORE
qualifier to indicate the time attribute to be used as the basis for selection:
/BACKUP, /CREATED (default), /EXPIRED, or /MODIFIED.

For complete information on specifying time values, see the *VMS DCL
Concepts Manual*.

/BY_OWNER[=uic]
Selects only those files whose owner user identification code (UIC) matches
the specified owner UIC. The default UIC is that of the current process.

Specify the UIC by using standard UIC format as described in the *VMS
DCL Concepts Manual*.

/CONFIRM
/NOCONFIRM (default)

Controls whether a request is issued before each delete operation to confirm that the operation should be performed on that file. The following responses are valid:

YES	NO	QUIT
TRUE	FALSE	Ctrl/Z
1	0	ALL
Return		

You can use any combination of uppercase and lowercase letters for word responses. Word responses can be abbreviated to one or more letters (for example, T, TR, or TRU for TRUE), but these abbreviations must be unique. Affirmative answers are YES, TRUE, and 1. Negative answers include: NO, FALSE, 0, and pressing the Return key. Entering QUIT or pressing Ctrl/Z indicates that you want to stop processing the command at that point. When you respond by entering ALL, the command continues to process, but no further prompts are given. If you type a response other than one of those in the list, DCL issues an error message and redisplays the prompt.

/CREATED (default)

Modifies the time value specified with the /BEFORE or the /SINCE qualifier. The /CREATED qualifier selects files based on their dates of creation. This qualifier is incompatible with the /BACKUP, /EXPIRED, and /MODIFIED qualifiers, which also allow you to select files according to time attributes. If you specify none of these four time qualifiers, the default is the /CREATED qualifier.

/ERASE
/NOERASE (default)

When you delete a file, the area in which the file was stored is returned to the system for future use. The data that was stored in that location still exists in the system until new data is written over it. When you specify the /ERASE qualifier, the storage location is overwritten with a system specified pattern so that the data no longer exists.

/EXCLUDE=(filespec[,...])

Excludes the specified files from the delete operation. You can include a directory but not a device in the file specification. Wildcard characters (* and %) are allowed in the file specification. However, you cannot use relative version numbers to exclude a specific version. If you specify only one file, you can omit the parentheses.

/EXPIRED

Modifies the time value specified with the /BEFORE or the /SINCE qualifier. The /EXPIRED qualifier selects files according to their expiration dates. (The expiration date is set with the SET FILE/EXPIRATION_DATE command.) The /EXPIRED qualifier is incompatible with the /BACKUP, /CREATED, and /MODIFIED qualifiers, which also allow you to select files according to time attributes. If you specify none of these four time qualifiers, the default is the /CREATED qualifier.

/LOG
/NOLOG (default)
Controls whether the DELETE command displays the file specification of each file after its deletion.

/MODIFIED
Modifies the time value specified with the /BEFORE or the /SINCE qualifier. The /MODIFIED qualifier selects files according to the dates on which they were last modified. This qualifier is incompatible with the /BACKUP, /CREATED, and /EXPIRED qualifiers, which also allow you to select files according to time attributes. If you specify none of these four time modifiers, the default is the /CREATED qualifier.

/SINCE[=time]
Selects only those files dated after the specified time. You can specify time as an absolute time, as a combination of absolute and delta times, or as one of the following keywords: TODAY (default), TOMORROW, or YESTERDAY. Specify one of the following qualifiers with the /SINCE qualifier to indicate the time attribute to be used as the basis for selection: /BACKUP, /CREATED (default), /EXPIRED, or /MODIFIED.

For complete information on specifying time values, see the *VMS DCL Concepts Manual*.

EXAMPLES

1 $ DELETE   COMMON.SUM;2

> The DELETE command deletes the file COMMON.SUM;2 from the current default disk and directory.

2 $ DELETE *.OLD;*

> The DELETE command deletes all versions of files with file type OLD from the default disk directory.

3 $ DELETE ALPHA.TXT;*, BETA;*, GAMMA;*

> The DELETE command deletes all versions of the files ALPHA.TXT, BETA.TXT, and GAMMA.TXT. The command uses the file type of the first input file as a temporary default. Note, however, that some form of version number (here specified as wildcards) must be included in each file specification.

4
```
$ DELETE /BEFORE=15-APR/LOG  *.DAT;*
%DELETE-I-FILDEL, DISK2:[MALCOLM]ASSIGN.DAT;1 deleted (5 block)
%DELETE-I-FILDEL, DISK2:[MALCOLM]BATCHAVE.DAT;3 deleted (4 blocks)
%DELETE-I-FILDEL, DISK2:[MALCOLM]BATCHAVE.DAT;2 deleted (4 blocks)
%DELETE-I-FILDEL, DISK2:[MALCOLM]BATCHAVE.DAT;1 deleted (4 blocks)
%DELETE-I-FILDEL, DISK2:[MALCOLM]CANCEL.DAT;1 deleted (2 blocks)
%DELETE-I-FILDEL, DISK2:[MALCOLM]DEFINE.DAT;1 deleted (3 blocks)
%DELETE-I-FILDEL, DISK2:[MALCOLM]EXIT.DAT;1 deleted (1 block)
%DELETE-I-TOTAL, 7 files deleted (23 blocks)
```

> The DELETE command deletes all versions of all files with file type DAT that were either created or updated before April 15 of this year. The /LOG

qualifier not only displays the name of each file deleted, but also the total number of files deleted.

5   `$ DELETE A.B;`

The DELETE command deletes the file A.B with the highest version number.

6
```
$ DELETE/CONFIRM/SINCE=TODAY [MALCOLM.TESTFILES]*.OBJ;*
DISK0:[MALCOLM.TESTFILES]AVERAG.OBJ;1, delete? [N]:Y
DISK0:[MALCOLM.TESTFILES]SCANLINE.OBJ;4, delete? [N]:N
DISK0:[MALCOLM.TESTFILES]SCANLINE.OBJ;3, delete? [N]:N
DISK0:[MALCOLM.TESTFILES]SCANLINE.OBJ;2, delete? [N]:N
DISK0:[MALCOLM.TESTFILES]WEATHER.OBJ;3, delete? [N]:Y
```

The DELETE command examines all versions of files with file type OBJ in the subdirectory [MALCOLM.TESTFILES], and locates those that were created or modified today. Before deleting each file, it requests confirmation that the file should be deleted. The default response—N—is given in brackets.

7
```
$ DIRECTORY [.SUBTEST]
%DIRECT-W-NOFILES, no files found
$ SET PROTECTION SUBTEST.DIR/PROTECTION=OWNER:D
$ DELETE SUBTEST.DIR;1
```

Before the directory file SUBTEST.DIR is deleted, the DIRECTORY command is used to verify that there are no files cataloged in the directory. The SET PROTECTION command redefines the protection for the directory file so that it can be deleted; then the DELETE command deletes it.

8   `$ DELETE DALLAS"THOMAS SECRET"::DISK0:[000,000]DECODE.LIS;1`

This DELETE command deletes the file DECODE.LIS;1 from the directory [000,000] on device DISK0 at remote node DALLAS. The user name and password follow the remote node name.

9
```
$ DELETE QUEBEC::"DISK1:DEAL.BIG"
$ DELETE QUEBEC::DISK1:DEAL.BIG;
```

Either of these DELETE commands can be used to delete the file DEAL.BIG on device ZZZ1 at remote node QUEBEC. Note that the DELETE command requires an explicit version number in a file specification, but the file to be deleted is on a remote node whose file syntax does not recognize version numbers. (QUEBEC is an RT–11 node.) Therefore, the file specification must either be enclosed in quotation marks (" ") or entered with a null version number (that is, a trailing semicolon [;]).

DELETE/CHARACTERISTIC

Deletes the definition of a queue characteristic previously established with the DEFINE/CHARACTERISTIC command. The /CHARACTERISTIC qualifier is required.

Requires OPER (operator) privilege.

FORMAT   **DELETE/CHARACTERISTIC** *characteristic-name*

PARAMETER   *characteristic-name*
Specifies the name of the characteristic to be deleted.

DESCRIPTION   The DELETE/CHARACTERISTIC command deletes a characteristic from the system characteristic table.

To change the number of an existing characteristic, you can use the DEFINE/CHARACTERISTIC command. It is not necessary to delete the characteristic before changing it.

QUALIFIER   */LOG*
/NOLOG (default)
Controls whether the DELETE/CHARACTERISTIC command displays the name of each characteristic after its deletion.

EXAMPLE

```
$  DEFINE/CHARACTERISTIC BLUE 7
   .
   .
   .
$  DELETE/CHARACTERISTIC BLUE
$  DEFINE/CHARACTERISTIC BLUE_INK 7
```

The DEFINE/CHARACTERISTIC command in this example establishes the characteristic BLUE, with number 7, to mean blue ink ribbons for printers. To change the name of the characteristic, enter the DELETE/CHARACTERISTIC command. Then enter another DEFINE /CHARACTERISTIC command to rename the characteristic to BLUE_ INK, using the characteristic number 7.

DELETE/ENTRY

Deletes one or more print or batch jobs. The jobs can be in progress or waiting in the queue. The /ENTRY qualifier is required.

Requires OPER (operator) privilege, execute (E) access to the queue, or delete (D) access to the specified jobs.

FORMAT

DELETE/ENTRY=*(entry-number[,...]) [queue-name[:]]*

PARAMETERS

entry-number[,...]

Specifies the entry number (or a list of entry numbers) of jobs to be deleted. If you specify only one entry number, you can omit the parentheses. If you do not specify a queue name, you can delete entries from multiple queues.

The system assigns a unique entry number to each queued print or batch job in the system. By default, the PRINT and SUBMIT commands display the entry number when they successfully queue a job for processing. These commands also create or update the local symbol $ENTRY to reflect the entry number of the most recently queued job. To find a job's entry number, enter the SHOW ENTRY or SHOW QUEUE command.

queue-name[:]

Specifies the name of the queue where the jobs are located. The queue name can refer either to the queue to which the job was submitted or to the queue where the job is executing. The *queue-name* parameter is optional syntax. However, when you specify a queue name, the VMS operating system uses it to verify an entry in the specific queue before deleting the entry.

DESCRIPTION

The DELETE/ENTRY command deletes one or more jobs from a queue. If you specify a queue name and more than one entry number with a DELETE/ENTRY command, all the jobs must be located in the same queue.

You can delete jobs that are currently executing, as well as jobs that are in other states. For example, DELETE/ENTRY can delete a job that is currently in a holding or a pending state.

QUALIFIER

/LOG
/NOLOG (default)

Controls whether the DELETE/ENTRY command displays the entry number of each batch or print job that it deletes.

EXAMPLES

1   `$ PRINT/HOLD  ALPHA.TXT`
`Job ALPHA (queue SYS$PRINT, entry 110) holding`

.
.
.

`$ DELETE/ENTRY=110  SYS$PRINT`

The PRINT command in this example queues a copy of the file
ALPHA.TXT in a HOLD status, to defer its printing until a SET ENTRY
/RELEASE command is entered. The system displays the job name, the
entry number, the name of the queue in which the job was entered, and
the status. Later, the DELETE/ENTRY command requests that the entry
be deleted from the queue SYS$PRINT.

2   `$ SUBMIT/AFTER=18:00  WEATHER`
`Job WEATHER (queue SYS$BATCH, entry 203) holding until 19-APR-1990 18:00`
`$ SUBMIT/HOLD/PARAMETERS=SCANLINE  DOFOR`
`Job DOFOR (queue SYS$BATCH, entry 210) holding`

.
.
.

`$ DELETE/ENTRY=(203,210)/LOG`
`%DELETE-W-SEARCHFAIL, error searching for 203`
`-JBC-E-NOSUCHENT, no such entry`
`%DELETE-I-DELETED, entry 210 aborting or deleted`

The SUBMIT commands in this example queue the command procedures
WEATHER.COM and DOFOR.COM for processing as batch jobs.
WEATHER.COM is queued for execution after 6:00 P.M. DOFOR.COM
is queued in a HOLD status and cannot execute until you enter a SET
ENTRY/RELEASE command. Later, the DELETE/ENTRY/LOG command
requests that the system delete both these entries from the queue and
display a message indicating that the entries have been deleted.

The job WEATHER (entry 203) has completed by the time the DELETE
/ENTRY/LOG command is entered. Thus, entry 203 no longer exists. Note
that a message indicates that there is no entry 203 in the queue. The job
DOFOR (entry 210) is in a HOLD status when the DELETE/ENTRY/LOG
command is entered. Thus, the system deletes entry 210 from the queue
and displays a message to that effect.

3   `$ PRINT CHAPTER8.MEM`
`Job CHAPTER8 (queue SYS$PRINT, entry 25) pending on queue SYS$PRINT`

.
.
.

`$ SHOW QUEUE SYS$PRINT`
`Printer queue SYS$PRINT, on PARROT::PARROT$LPA0, mounted form DEFAULT`

Jobname	Username	Entry	Blocks	Status
CHAPTER7	SMITH	24	274	Pending
CHAPTER8	SMITH	25	976	Pending

`$ DELETE/ENTRY=25`

The PRINT command in this example submits the file CHAPTER8.MEM

to the printer queue SYS$PRINT. Later, user Smith needs to edit the file again before printing it. Using the SHOW QUEUE command, Smith verifies that the job is still pending and that the entry number for the job is 25. Smith then enters the DELETE/ENTRY command to delete the job from the queue.

DELETE/FORM

Deletes a form (for printer or terminal queues) previously established with the DEFINE/FORM command. The /FORM qualifier is required.

Requires OPER (operator) privilege.

FORMAT   **DELETE/FORM** *form-name*

PARAMETER   ***form-name***
Specifies the name of the form to be deleted.

DESCRIPTION The DELETE/FORM command deletes a form definition from the system forms table. When you delete a form, there can be no outstanding references to the form either in queues that have been mounted with the form or by jobs requesting that form. To locate all references to the form, use the SHOW QUEUE/FULL command.

To change the number or attributes of an existing form, use the DEFINE /FORM command. It is not necessary to delete a form before changing it.

QUALIFIER   ***/LOG***
/NOLOG (default)
Controls whether the DELETE/FORM command displays the name of each form after its deletion.

EXAMPLES

1 $ DELETE/FORM CENTER

The DELETE/FORM command in this example deletes the form named CENTER.

2 $ DEFINE/FORM /DESCRIPTION="letter size continuous form paper" CFLET 7
        .
        .
        .
    $ DELETE/FORM CFLET
    $ DEFINE/FORM /DESCRIPTION="letter size continuous form paper" LETTER_CONT 7

The DEFINE/FORM command in this example establishes the form CFLET with number 7 to mean continuous-form paper 8.5 inches by 11 inches. To change the name of the form, delete the form named CFLET and define a new one named LETTER_CONT.

DELETE/INTRUSION_RECORD

Removes an entry from the break-in database.

Requires CMKRNL (change mode to kernel) and SECURITY privileges.

FORMAT

DELETE/INTRUSION_RECORD *source*

PARAMETER

source

Specifies the source field of the entry to be removed from the break-in database.

DESCRIPTION

Use the DELETE/INTRUSION_RECORD command to remove an entry from the break-in database. For example, if the user Hammer repeatedly attempted to log in to terminal TTA24 with an expired password, the SHOW INTRUSION command would display the following entry:

```
Intrusion   Type       Count    Expiration    Source

TERM_USER   INTRUDER   9        10:29:39.16   TTA24:HAMMER
```

The terminal is locked out of the system because the login failure limit has been reached. When Hammer approaches you and you identify the problem as an expired password, you can then use the DELETE /INTRUSION command to remove the record from the break-in database.

EXAMPLES

1 $ DELETE/INTRUSION_RECORD TTC2:

In this example, the DELETE/INTRUSION_RECORD command removes all intrusion records generated by break-in attempts on TTC2. No username is specified because none of the login failures occurred for valid users.

2 $ DELETE/INTRUSION_RECORD NODE1::HAMMER

This command removes all intrusion entries generated from node NODE1 for user HAMMER.

DELETE/KEY

Deletes key definitions that have been established by the DEFINE/KEY command. The /KEY qualifier is required.

FORMAT   **DELETE/KEY**   *[key-name]*

PARAMETER   *key-name*
Specifies the name of the key to be deleted. This parameter is incompatible with the /ALL qualifier.

QUALIFIERS   */ALL*
Deletes all key definitions in the specified state; the default is the current state. If you use the /ALL qualifier, do not specify a key name. Use the /STATE qualifier to specify one or more states.

/LOG (default)
/NOLOG
Controls whether messages are displayed indicating that the specified key definitions have been deleted.

/STATE=(state-name[,...])
/NOSTATE (default)
Specifies the name of the state for which the specified key definition is to be deleted. The default state is the current state. If you specify only one state name, you can omit the parentheses. State names can be any alphanumeric string.

EXAMPLES

1 $ DELETE/KEY/ALL
     %DCL-I-DELKEY, DEFAULT key PF1 has been deleted
     %DCL-I-DELKEY, DEFAULT key PF2 has been deleted
     %DCL-I-DELKEY, DEFAULT key PF3 has been deleted
     %DCL-I-DELKEY, DEFAULT key PF4 has been deleted
     $

In this example, the user has defined keys PF1 to PF4 in the default state. The DELETE/KEY command deletes all key definitions in the current state, which is the default state.

```
2   $ DEFINE/KEY PF3 "SHOW TIME" /TERMINATE
    %DCL-I-DEFKEY, DEFAULT key PF3 has been defined
    $ PF3
    $ SHOW TIME
     19-APR-1990 14:43:59
        .
        .
        .

    $ DELETE/KEY PF3
    %/DCL-I-DELKEY, DEFAULT key PF3 has been deleted
    $ PF3
    $
```

> In this example, the DEFINE/KEY command defines the PF3 key on the
> keypad as SHOW TIME. To delete the definition for the PF3 key, use the
> DELETE/KEY command. When the user presses PF3, only the system
> prompt is displayed.

DELETE/QUEUE

Deletes a print or batch queue created by the INITIALIZE/QUEUE command, and deletes all the jobs in the queue. The /QUEUE qualifier is required.

Requires OPER (operator) privilege.

FORMAT	**DELETE/QUEUE** *queue-name[:]*

PARAMETER	***queue-name[:]*** Specifies the name of the queue to be deleted.

DESCRIPTION   To delete a queue, use the following procedure:

1   Stop the specified queue by using the STOP/QUEUE/NEXT command.

   The STOP/QUEUE/NEXT command stops the specified queue after all executing jobs have completed processing. Wait for any executing jobs to complete processing.

2   Make sure that there are no outstanding references to the specified queue.

   If a generic queue refers to the specified queue as a target execution queue, you must remove the specified queue from the list of target execution queues.

   If a logical queue refers to the specified queue, you must deassign the logical queue.

   If the specified queue is a generic queue, jobs that were entered initially on the generic queue and are executing on any of its target queues count as references to the specified queue. Before you can delete the specified queue, either you must delete any jobs that were submitted originally to the specified queue and are executing on its target queues, or you must wait until these jobs have completed processing.

3   To move jobs from the specified queue to another queue, use the SET ENTRY/REQUEUE or ASSIGN/MERGE commands. Any jobs that remain in the specified queue are deleted when the queue is deleted.

4   Enter the DELETE/QUEUE command.

QUALIFIER   ***/LOG***
/NOLOG (default)
Controls whether the DELETE/QUEUE command displays the name of each queue after it is deleted.

EXAMPLE

```
$  INITIALIZE/QUEUE/DEFAULT=FLAG/START/ON=LPA0 LPA0_QUEUE
   .
   .
   .
$  STOP/QUEUE/NEXT LPA0_QUEUE
$  DELETE/QUEUE LPA0_QUEUE
```

In this example, the first command initializes and starts the printer queue LPA0_QUEUE. The STOP/QUEUE/NEXT command stops the queue. The DELETE/QUEUE command deletes the queue.

DELETE/SYMBOL

Deletes one or all symbol definitions from a local or global symbol table. The /SYMBOL qualifier is required.

FORMAT	**DELETE/SYMBOL** *[symbol-name]*

PARAMETER   *symbol-name*

Specifies the name of the symbol to be deleted. A name is required unless the /ALL qualifier is specified. The *symbol-name* parameter is incompatible with the /ALL qualifier. Symbol names can have from 1 to 255 characters. By default, the DELETE/SYMBOL command assumes that the symbol is in the local symbol table for the current command procedure.

DESCRIPTION   The DELETE/SYMBOL command deletes a symbol definition from a symbol table. If you do not specify either the global or local symbol table, the symbol is deleted from the local table. If you specify both the /GLOBAL and /LOCAL qualifiers, only the last specified qualifier is accepted. The /SYMBOL qualifier must always immediately follow the DELETE command name.

QUALIFIERS   */ALL*

Deletes all symbols from the specified table. If you do not specify either the /LOCAL or the /GLOBAL qualifier, all symbols defined at the current command level are deleted. The /ALL qualifier is incompatible with the *symbol-name* parameter.

/GLOBAL

Deletes the symbol from the global symbol table of the current process.

/LOCAL (default)

Deletes the symbol from the local symbol table of the current process.

/LOG
/NOLOG (default)

Controls whether an informational message listing each symbol being deleted is displayed.

EXAMPLES

1   $ DELETE/SYMBOL/ALL

In this example, the DELETE/SYMBOL command deletes all symbol definitions at the current command level.

2   `$ DELETE/SYMBOL/LOG FOO`
      `%DCL-I-DELSYM, LOCAL symbol FOO has been deleted`

> In this example, the DELETE/SYMBOL command deletes the symbol FOO from the local symbol table for the current process. In addition, the /LOG qualifier causes an informational message, listing the symbol being deleted, to be displayed.

3   `$ DELETE/SYMBOL/GLOBAL PDEL`

> In this example, the DELETE/SYMBOL command deletes the symbol named PDEL from the global symbol table for the current process.

DEPOSIT

Replaces the contents of the specified locations in virtual memory and displays the new contents.

The DEPOSIT command, together with the EXAMINE command, aids in debugging programs interactively. The DCL command DEPOSIT is similar to the DEPOSIT command of the VMS Symbolic Debugger.

Requires user-mode read (R) and write (W) access to the virtual memory location whose contents you wish to change.

FORMAT   DEPOSIT   *location=data[,...]*

PARAMETERS   *location*
Specifies the starting virtual address or range of virtual addresses (where the second address is larger than the first) whose contents are to be changed. A location can be any valid integer expression containing an integer value, a symbol name, a lexical function, or a combination of these entities. Radix qualifiers determine the radix in which the address is interpreted; hexadecimal is the initial default radix. Symbol names are always interpreted in the radix in which they were defined. The radix operators %X, %D, or %O can precede the location. A hexadecimal value must begin with a number (or be preceded by %X).

The specified location must be within the virtual address space of the image currently running in the process.

The DEPOSIT and EXAMINE commands maintain a pointer to a current memory location. The DEPOSIT command sets this pointer to the byte following the last byte modified; you can refer to this pointer by using a period (.) in subsequent EXAMINE and DEPOSIT commands. If the DEPOSIT command cannot deposit the specified data, the pointer does not change. The EXAMINE command does not change the value of the pointer.

data[,...]
Specifies the data to be deposited into the specified locations. By default, the data is assumed to be in hexadecimal format; it is then converted to binary format and is written into the specified location.

If you specify more than one item, separate the items with commas (,). The DEPOSIT command writes the data in consecutive locations, beginning with the address specified.

When non-ASCII data is deposited, you can specify each item of data using any valid integer expression.

When ASCII data is deposited, only one item of data is allowed. All characters to the right of the equal sign are considered to be part of a single string. The characters are converted to uppercase, and all spaces are compressed.

DEPOSIT

DESCRIPTION
When the DEPOSIT command completes, it displays both the virtual memory address into which data is deposited and the new contents of the location, as follows:

```
address:  contents
```

If the specified address can be read from but not written to by the current access mode, the DEPOSIT command displays the original contents of the location. If the specified address can be neither read from nor written to, the DEPOSIT command displays asterisks (*) in the data field. The DEPOSIT command maintains a pointer at that location (at the byte following the last byte modified).

If you specify a list of numeric values, some but not all of the values may be successfully deposited before an access violation occurs. If an access violation occurs while ASCII data is being deposited, nothing is deposited.

Radix Qualifiers: The radix default for a DEPOSIT or EXAMINE command determines how the command interpreter interprets numeric literals. The initial default radix is hexadecimal; all numeric literals in the command line are assumed to be hexadecimal values. If a radix qualifier modifies the command, that radix becomes the default for subsequent EXAMINE and DEPOSIT commands, until another qualifier overrides it. For example:

```
$ DEPOSIT/DECIMAL 900=256
00000384:  256
```

The DEPOSIT command interprets both the location 900 and the value 256 as decimal. All subsequent DEPOSIT and EXAMINE commands assume that numbers you enter for addresses and data are decimal. Note that the DEPOSIT command always displays the address location in hexadecimal.

Symbol values defined by = (assignment statement) commands are always interpreted in the radix in which they were defined.

Note that hexadecimal values entered as deposit locations or as data to be deposited must begin with a numeric character (0 to 9). Otherwise, the command interpreter assumes that you have entered a symbol name and attempts symbol substitution.

You can use the radix operators %X, %D, or %O to override the current default when you enter the DEPOSIT command. For example:

```
$ DEPOSIT/DECIMAL %X900=10
```

This command deposits the decimal value 10 in the location specified as hexadecimal 900.

Length Qualifiers: The initial default length unit for the DEPOSIT command is a longword. If a list of data values is specified, the data is deposited into consecutive longwords beginning at the specified location. If a length qualifier modifies the command, that length becomes the default for subsequent EXAMINE and DEPOSIT commands, until another qualifier overrides it. If you specify data values that are longer than the specified length, an error occurs.

Length qualifiers are ignored when ASCII values are deposited.

Restriction on Placement of Qualifiers: The DEPOSIT command
analyzes expressions arithmetically. Therefore, qualifiers, which must
be preceded by a slash (/), must appear immediately after the command
name to be interpreted correctly.

QUALIFIERS	**/ASCII**

/ASCII
Indicates that the specified data is ASCII.

Only one data item is allowed; all characters to the right of the equal sign
(=) are considered to be part of a single string. Unless they are enclosed
within quotation marks (" "), characters are converted to uppercase and
multiple spaces are compressed to a single space before the data is written
in memory.

The DEPOSIT command converts the data to its binary equivalent before
placing it in virtual memory. When you specify /ASCII, or when ASCII
mode is the default, the location you specify is assumed to be hexadecimal.

/BYTE
Requests that data be deposited 1 byte at a time.

/DECIMAL
Indicates that the data is decimal. The DEPOSIT command converts the
data to its binary equivalent before placing it in virtual memory.

/HEXADECIMAL
Indicates that the data is hexadecimal. The DEPOSIT command converts
the data to its binary equivalent before placing it in virtual memory.

/LONGWORD
Requests that data be deposited a longword at a time.

/OCTAL
Indicates that the data is octal. The DEPOSIT command converts the data
to its binary equivalent before placing it in virtual memory.

/WORD
Requests that the data be deposited one word at a time.

EXAMPLES

1
```
$ RUN MYPROG
    .
    .
    .

Ctrl/Y
$ EXAMINE 2780
00002780:  1C50B344
$ DEPOSIT .=0
00002780:  00000000
$ CONTINUE
```

The RUN command executes the image MYPROG.EXE; subsequently,
Ctrl/Y interrupts the program. Assuming that the initial defaults of the

DEPOSIT

/HEXADECIMAL and /LONGWORD qualifiers are in effect, the DEPOSIT command places a longword of zeros in virtual memory location 2780.

Because the EXAMINE command sets up a pointer to the current memory location, which in this case is virtual address 2780, you can refer to this location with a period (.) in the DEPOSIT command.

The CONTINUE command resumes execution of the image.

2
```
$ DEPOSIT/ASCII   2C00=FILE: NAME: TYPE:
00002C00:  FILE: NAME: TYPE:...
```

In this example, the DEPOSIT command deposits character data at hexadecimal location 2C00 and displays the contents of the location after modifying it. Because the current default length is a longword, the response from the DEPOSIT command displays full longwords. The ellipsis (...) indicates that the remainder of the last longword of data contains information that was not modified by the DEPOSIT command.

3
```
$ EXAMINE 9C0          ! Look at Hex location 9C0
000009C0:  8C037DB3
$ DEPOSIT .=0          ! Deposit longword of 0
000009C0:  00000000
$ DEPOSIT/BYTE .=1     ! Put 1 byte at next location
000009C4:  01
$ DEPOSIT .+2=55       ! Deposit 55 next
000009C7:  55
$ DEPOSIT/LONG .=0C,0D,0E ! Deposit longwords
000009C8:  0000000C 0000000D 0000000E
```

The sequence of DEPOSIT commands in the above example illustrates how the DEPOSIT command changes the current position pointer. Note that after you specify the /BYTE qualifier, all data is deposited and displayed in bytes, until the /LONGWORD qualifier restores the system default.

4
```
$ BASE=%X200           ! Define a base address
$ LIST=BASE+%X40       ! Define offset from base
$ DEPOSIT/DECIMAL LIST=1,22,333,4444
00000240:  00000001 00000022 00000333 00004444
$ EXAMINE/HEX LIST:LIST+0C ! Display results in hex
00000240:  00000001 00000016 0000014D 0000115C
```

The assignment statements define a base address in hexadecimal and a label at a hexadecimal offset from the base address. The DEPOSIT command reads the list of values and deposits each value into a longword, beginning at the specified location. The EXAMINE command requests a hexadecimal display of these values.

DIFFERENCES

Compares the contents of two disk files and displays a listing of the records that do not match.

FORMAT   **DIFFERENCES**   *input1-filespec [input2-filespec]*

PARAMETERS   *input1-filespec*
Specifies the first file to be compared. The file specification must include a file name and a file type. Wildcard characters are not allowed.

input2-filespec
Specifies the second file to be compared. Unspecified fields default to the corresponding fields in the *input1-filespec* parameter. Wildcard characters are not allowed.

If you do not specify a secondary input file, the DIFFERENCES command uses the next lower version of the primary input file.

DESCRIPTION   Use the DIFFERENCES command to determine whether two files are identical and, if not, how they differ. The DIFFERENCES command compares the two specified files on a record-by-record basis and produces an output file that lists the DIFFERENCES, if any.

The qualifiers for the DIFFERENCES command can be categorized according to their functions, as follows:

- Qualifiers that request the DIFFERENCES command to ignore data in each record:

    /COMMENT_DELIMITERS
    /IGNORE

These qualifiers allow you to define characters that denote comments or to designate characters or classes of characters to ignore when comparing files. For example, you can have the DIFFERENCES command ignore extra blank lines or extra spaces within lines.

By default, the DIFFERENCES command compares every character in each record.

- Qualifiers that control the format of the information contained in the list of differences:

    /CHANGE_BAR
    /IGNORE
    /MERGED
    /MODE
    /PARALLEL

DIFFERENCES

/SEPARATED
/SLP
/WIDTH

By default, the DIFFERENCES command merges the differences it finds in the files being compared. It lists each record in the file that has no match in the other input file and then lists the next record that it finds that does have a match.

By default, the DIFFERENCES command also supplies a line number with each listed record, and it lists the records with all designated ignore characters deleted.

You can specify combinations of qualifiers to request an output listing that includes the comparison in more than one format. Note that SLP output is incompatible with all other types of output; parallel output can be generated only in ASCII mode.

- Qualifiers that control the extent of the comparison:

/MATCH
/MAXIMUM_DIFFERENCES
/WINDOW

By default, the DIFFERENCES command reads every record in the master input file and looks for a matching record in the revision input file. A search for a match between the two input files continues until either a match is found or the ends of the two files are reached. Sections of the two files are considered a match only if three sequential records are found to be identical in each file.

By default, DIFFERENCES command output is written to the current SYS$OUTPUT device. Use the /OUTPUT qualifier to request that the DIFFERENCES command write the output to an alternate file or device.

The DIFFERENCES command terminates with an exit status. The following severity levels indicate the result of the comparison:

SUCCESS	Files are identical.
INFORMATIONAL	Files are different.
WARNING	User-specified maximum number of DIFFERENCES has been exceeded.
ERROR	Insufficient virtual memory to complete comparison.

All severity levels other than SUCCESS indicate that the two input files are different.

QUALIFIERS

/CHANGE_BAR[=([change-char][,[NO]NUMBER])]

Marks with the specified character in the left margin each line in the input1 file that differs from the corresponding line in the input2 file. If you do not specify a change bar character, the default is an exclamation point (!) for ASCII output. If you specify hexadecimal or octal output (see the description of the /MODE qualifier), the change bar character is ignored and differences are marked by a "***CHANGE***" string in the record header. The keyword NONUMBER suppresses line numbers in the listing. If neither the NUMBER nor the NONUMBER keyword is specified, the

default is controlled by the /[NO]NUMBER command qualifier. If you specify only one option, you can omit the parentheses. If you use an exclamation point (!) as the specified character, you must enclose it in quotation marks (" "); for example, /CHANGE_BAR=("!",NUMBER).

/COMMENT_DELIMITER[=(character[,...])]

Ignores lines starting with a specified comment character. If the comment character is an exclamation point or semicolon (;), it can appear anywhere in the line and characters to the right of the character are ignored. If you specify just one character, you can omit the parentheses. Lowercase characters are automatically converted to uppercase unless they are enclosed in quotation marks. Nonalphanumeric characters (such as ! and ,) must be enclosed in quotation marks. You can specify up to 32 comment characters by typing the character itself or one of the following keywords. (Keywords can be abbreviated provided that the resultant keyword is not ambiguous and has at least 2 characters; single letters are treated as delimiters.)

Keyword	Character
COLON	Colon (:)
COMMA	Comma (,)
EXCLAMATION	Exclamation point (!)
FORM_FEED	Form feed
LEFT	Left bracket ([ )
RIGHT	Right bracket ( ])
SEMI_COLON	Semicolon (;)
SLASH	Slash (/)
SPACE	Space
TAB	Tab

The /COMMENT_DELIMITER qualifier is used with or without the /IGNORE=COMMENTS qualifier to indicate which comments are to be ignored.

If both the uppercase and lowercase forms of a letter are to be used as comment characters, the letter must be specified twice, once in uppercase and once in lowercase. If you do not include either a comment character or a keyword with the /COMMENT_DELIMITER qualifier, the DIFFERENCES command assumes a default comment character based on the file type. For some file types (COB and FOR), the default comment characters are considered valid delimiters only if they appear in the first column of a line. Multicharacter comment characters are not allowed.

The following characters are the default comment delimiters for files with the specified file types:

File Type	Default Comment Character
B2S, B32, BAS, BLI	!
CBL, CMD	! and ;

File Type	Default Comment Character
COB	* or / in the first column
COM, COR	!
FOR	! anywhere and C, D, c, d in the first column
HLP	!
MAC, MAR	;
R32, REQ	!

/IGNORE=(keyword[,...])

Inhibits the comparison of the specified characters, strings, or records; also controls whether the comparison records are output to the listing file as edited records or exactly as they appeared in the input file. If you specify only one keyword, you can omit the parentheses. The keyword parameter refers to either a character or a keyword. The first set of keywords determines what, if anything, is ignored during file comparison; the second set of keywords determines whether or not ignored characters are included in the output. The following keywords are valid options for the /IGNORE qualifier:

Keyword	Item Ignored
BLANK_LINES	Blank lines between data lines.
COMMENTS	Data following a comment character. (Use the /COMMENT_DELIMITER qualifier to designate one or more nondefault comment delimiters.)
FORM_FEEDS	Form feed character.
HEADER[=n]	First n records of the file, beginning with a record whose first character is a form feed. The first record is not ignored if the only character it contains is a form feed. (N indicates the number of records and defaults to 2. A record with a single form feed is not counted.)
SPACING	Extra blank spaces or tabs within data lines.
TRAILING_SPACES	Space and tab characters at the end of a data line.

Keyword	Status of Ignored Items in Output
EDITED	Omits ignored characters from the output records.
EXACT	Includes ignored characters in the output records.
PRETTY	Formats output records.

Each data line is checked for COMMENTS, FORM_FEEDS, HEADER, and SPACING before it is tested for TRAILING_SPACES and then BLANK_LINES. Therefore, if you direct the DIFFERENCES command to ignore COMMENTS, TRAILING_SPACES, and BLANK_LINES, it ignores a record that contains several spaces or blank lines followed by a comment.

By default, the DIFFERENCES command compares every character in each file and reports all differences. Also, by default, the DIFFERENCES command lists records in the output file with all ignored characters deleted.

If you specify the /PARALLEL qualifier, output records are always formatted. To format output records, specify the following characters:

Character	Formatted Output
Tab (Ctrl/I)	1–8 spaces
Return (Ctrl/M)	<CR>
Line feed (Ctrl/J)	<LF>
Vertical tab (Ctrl/K)	<VT>
Form feed (Ctrl/L)	<FF>
Other nonprinting characters	. (period)

/MATCH=size
Specifies the number of records that should indicate matching data after a difference is found. By default, after the DIFFERENCES command finds unmatched records, it assumes that the files once again match after it finds three sequential records that match. Use the /MATCH qualifier to override the default match size of 3.

You can increase the /MATCH qualifier value if you feel that the DIFFERENCES command is incorrectly matching sections of the master and revision input files after it has detected a difference.

/MAXIMUM_DIFFERENCES=n
Terminates the DIFFERENCES command after the specified number of unmatched records (specified with the n parameter) is found.

The number of unmatched records is determined by finding the maximum number of difference records for each difference section and adding them together.

If the DIFFERENCES command reaches the maximum number of differences that you specify, it will output only those records that were detected before the maximum was reached. Also, it will output, at most, one listing format and return a warning message.

By default, there is no maximum number of differences. All records in the specified input files are compared.

/MERGED[=n]
Specifies that the output file contain a merged list of differences with the specified number of matched records listed after each group of unmatched records. The value of the parameter n must be less than or equal to the number specified in the /MATCH qualifier. By default, the DIFFERENCES command produces a merged listing with one matched record listed after each set of unmatched records (that is, /MERGED=1). If the /MERGED, the /SEPARATED, or the /PARALLEL qualifier is not specified, the resulting output is merged, with one matched record following each unmatched record.

Use the /MERGED qualifier to override the default value of the parameter n, or to include a merged listing with other types of output.

/MODE=(radix[,...])

Specifies the format of the output. You can request that the output be formatted in one or more radix modes by specifying the following keywords, which may be abbreviated: ASCII (default), HEXADECIMAL, or OCTAL. If you specify only one radix, you can omit the parentheses.

By default, the DIFFERENCES command writes the output file in ASCII. If you specify more than one radix, the output listing contains the file comparison in each specified radix. When you specify two or more radix modes, separate them with commas.

If you specify the /PARALLEL or the /SLP qualifier, the /MODE qualifier is ignored for that listing form.

/NUMBER (default)
/NONUMBER

Includes line numbers in the listing of DIFFERENCES.

/OUTPUT[=filespec]

Specifies an output file to receive the list of differences. By default, the output is written to the current SYS$OUTPUT device. If the *filespec* parameter is not specified, the output is directed to the first input file with a file type of DIF. No wildcard characters are allowed.

When you specify the /OUTPUT qualifier, you can control the defaults applied to the output file specification as described in the *VMS DCL Concepts Manual*. The default output file type is DIF.

/PARALLEL[=n]

Lists the records with differences side by side. The value of the parameter *n* specifies the number of matched records to merge after each unmatched record; it must be a non-negative decimal number less than or equal to the number specified in the /MATCH qualifier.

By default, the DIFFERENCES command does not list records after each list of unmatched records. Also by default, the DIFFERENCES command creates only a list of merged differences.

/SEPARATED[=(input1-filespec[,input2-filespec])]

Lists sequentially only the records from the specified file that contain differences. If no files are specified, a separate listing is generated for each file. If only one file is specified, you can omit the parentheses. To specify the *input1-filespec* parameter, use either the first input file specified as the DIFFERENCES command parameter or the keyword MASTER. To specify the *input2-filespec* parameter, use either the second input file specified as the DIFFERENCES command parameter or the keyword REVISION.

By default, the DIFFERENCES command creates only a merged list of differences.

/SLP

Requests that the DIFFERENCES command produce an output file suitable for input to the SLP editor. If you use the /SLP qualifier, you cannot specify any of the following output file qualifiers: /MERGED, /PARALLEL, /SEPARATED, or /CHANGE_BAR.

Use the output file produced by the SLP qualifier as input to SLP to update the master input file, that is, to make the master input file match the revision input file.

When you specify the /SLP qualifier and you do not specify the /OUTPUT qualifier, the DIFFERENCES command writes the output file to a file with the same file name as the master input file with the file type DIF.

/WIDTH=n

Specifies the width of the lines in the output file. The default is 132 characters. If output is written to the terminal, the /WIDTH qualifier is ignored and the terminal line width is used.

Use the SET TERMINAL command to change the terminal line width.

/WINDOW=size

Searches the number of records specified by the *size* parameter, before a record is declared as unmatched. By default, the DIFFERENCES command searches to the ends of both input files before listing a record as unmatched.

The window size is the minimum size of a differences section that will cause the DIFFERENCES command to lose synchronization between the two input files.

EXAMPLES

```
1    $ DIFFERENCES EXAMPLE.TXT
     ************
     File DISK1:[GEORGE.TEXT]EXAMPLE.TXT;2
          1    DEMONSTRATION
          2    OF V3.0 DIFFERENCES
          3    UTILITY
     ******
     File DISK1:[GEORGE.TEXT]EXAMPLE.TXT;1
          1    DEMONSTRETION
          2    OF VMS DIFFERENCES
          3    UTILITY
     ************
     Number of difference sections found: 1
     Number of difference records found: 2
     DIFFERENCES/MERGED=1-
             DISK1:[GEORGE.TEXT]EXAMPLE.TXT;2
             DISK1:[GEORGE.TEXT]EXAMPLE.TXT;1
```

In this example, the DIFFERENCES command compares the contents of the two most recent versions of the file EXAMPLE.TXT in the current default directory. The DIFFERENCES command compares every character in every record and displays the results at the terminal.

DIFFERENCES

`$ DIFFERENCES/PARALLEL/WIDTH=80/COMMENT_DELIMITER="V" EXAMPLE.TXT`

```
-----------------------------------------------------------------------
File DISK1:[GEORGE.TEXT]EXAMPLE.TXT;2  |  File DISK1:[GEORGE.TEXT]EXAMPLE.TXT;1
------------------ 1 ----------------------------------------- 1 -----------------
DEMONSTRATION                          |  DEMONSTRETION
-----------------------------------------------------------------------
Number of difference sections found: 1
Number of difference records found: 1
DIFFERENCES/IGNORE=(COMMENTS)/COMMENT_DELIMITER=("V")/WIDTH=80/PARALLEL-
        DISK1:[GEORGE.TEXT]EXAMPLE.TXT;2-
        DISK1:[GEORGE.TEXT]EXAMPLE.TXT;1
```

> The DIFFERENCES command compares the same files as in Example 1, but ignores all comments following the first "V" encountered by the DIFFERENCES command. The command also specifies that an 80-column parallel list of differences be displayed.

`$ DIFFERENCES/WIDTH=80/MODE=(HEX,ASCII) EXAMPLE.TXT/CHANGE_BAR`

```
************
File DISK1:[GEORGE.TEXT]EXAMPLE.TXT;2
    1 ! DEMONSTRATION
    2 ! OF V3.0 DIFFERENCES
    3   UTILITY
************
************
File DISK1:[GEORGE.TEXT]EXAMPLE.TXT;2
RECORD NUMBER 1 (00000001) LENGTH 14 (0000000E)   ***CHANGE***
     204E 4F495441 5254534E 4F4D4544 DEMONSTRATION .. 000000
RECORD NUMBER 2 (00000002) LENGTH 19 (00000013)   ***CHANGE***
  4E455245 46464944 20302E33 5620464F OF V3.0 DIFFEREN 000000
                              534543 CES............. 000010
RECORD NUMBER 3 (00000003) LENGTH 7 (00000007)
                    595449 4C495455 UTILITY......... 000000
************
Number of difference sections found: 1
Number of difference records found: 2
DIFFERENCES /WIDTH=80/MODE=(HEX,ASCII)
        DISK1:[GEORGE.TEXT]EXAMPLE.TXT;2/CHANGE_BAR-
        DISK1:[GEORGE.TEXT]EXAMPLE.TXT;1
```

> The DIFFERENCES command compares the same files as in Example 1, but lists the differences in both hexadecimal and ASCII formats. The command also specifies that default change bars be used in the output. The default change bar notation for the hexadecimal output is ***CHANGE***. For the ASCII output, the default change bar character is the exclamation point.

`$ DIFFERENCES/OUTPUT BOSTON::DISK2:TEST.DAT OMAHA::DISK1:[PGM]TEST.DAT`

> The DIFFERENCES command compares two remote files and displays any differences found. The first file is TEST.DAT on remote node BOSTON. The second file is also named TEST.DAT on remote node OMAHA. The DIFFERENCES output is located in the file DISK1:[PGM]TEST.DIF.

DIRECTORY

Provides a list of files or information about a file or group of files.

Requires read (R) access to the directories to obtain any information. Requires read (R) access to the files to obtain information other than the file name.

FORMAT   **DIRECTORY**   *[filespec[,...]]*

PARAMETER   *filespec[,...]*
Specifies one or more files to be listed. The syntax of a file specification determines which files will be listed, as follows:

- If you do not enter a file specification, the DIRECTORY command lists all versions of the files in the current default directory.

- If you specify only a device name, the DIRECTORY command uses your default directory specification.

- Whenever the file specification does not include a file name, a file type, and a version number, all versions of all files in the specified directory are listed.

- If a file specification contains a file name or a file type, or both, and no version number, the DIRECTORY command lists all versions.

- If a file specification contains only a file name, the DIRECTORY command lists all files in the current default directory with that file type, regardless of file type and version number.

- If a file specification contains only a file type, the DIRECTORY command lists all files in the current default directory with that file type, regardless of file name and version number.

Wildcard characters can be used in the directory specification, file name, file type, or version number fields of a file specification to list all files that satisfy the components you specify. If you specify more than one file, separate the file specifications with either commas (,) or plus signs (+).

DESCRIPTION   The DIRECTORY command lists the files contained in a directory. When you use certain qualifiers with the command, additional information is displayed, along with the names of the files.

The output of the DIRECTORY command depends on certain formatting qualifiers and their defaults. These qualifiers are as follows: /COLUMNS, /DATE, /FULL, /OWNER, /PROTECTION, and /SIZE. However, the files are always listed in alphabetical order, with the highest numbered versions listed first.

DIRECTORY

In studying the qualifiers and the capabilities they offer, watch for qualifiers that work together and for qualifiers that override other qualifiers. For example, if you specify the /FULL qualifier, the system cannot display all the information in more than one column. Thus, if you specify both the /COLUMNS and /FULL qualifiers, the number of columns you request is ignored.

QUALIFIERS

/ACL
Controls whether the access control list (ACL) is displayed for each file. By default, the DIRECTORY command does not display the ACL for each file. The /ACL qualifier overrides the /COLUMNS qualifier.

/BACKUP
Modifies the time value specified with the /BEFORE or the /SINCE qualifier. The /BACKUP qualifier selects files according to the dates of their most recent backups. This qualifier is incompatible with the /CREATED, /EXPIRED, and /MODIFIED qualifiers, which also allow you to select files according to time attributes. If you specify none of these four time qualifiers, the default is the /CREATED qualifier.

/BEFORE[=time]
Selects only those files dated prior to the specified time. You can specify time as an absolute time, as a combination of absolute and delta times, or as one of the following keywords: TODAY (default), TOMORROW, or YESTERDAY. Specify one of the following qualifiers with the /BEFORE qualifier to indicate the time attribute to be used as the basis for selection: /BACKUP, /CREATED (default), /EXPIRED, or /MODIFIED.

For complete information on specifying time values, see the *VMS DCL Concepts Manual*.

/BRIEF (default)
Displays only a file's name, type, and version number. The brief format lists the files in alphabetical order from left to right on each line, in descending version number order. You can use the /ACL, /DATE, /FILE_ ID, /FULL, /NOHEADING, /OWNER, /PROTECTION, /SECURITY, and /SIZE qualifiers to expand a brief display.

/BY_OWNER[=uic]
Selects only those files whose owner user identification code (UIC) matches the specified owner UIC. The default UIC is that of the current process.

Specify the UIC by using standard UIC format as described in the *VMS DCL Concepts Manual*.

/COLUMNS=n
Specifies the number of columns in a brief display. The default is four. However, you can request as many columns as you like, restricted by the value of the /WIDTH qualifier. The /COLUMNS qualifier is incompatible with the /ACL, /FULL, and /SECURITY qualifiers.

The number of columns actually displayed depends on the amount of information requested for each column and the display value of the /WIDTH qualifier. The system displays only as many columns as can fit within the default or specified display width, regardless of how many columns you specify with the /COLUMNS qualifier.

The DIRECTORY command truncates long file names only when you have asked for additional information to be included in each column. The default file name size is 19 characters. Use the /WIDTH qualifier to change the default. When a file name is truncated, the system displays one less character than the file name field size and inserts a vertical bar in the last position. For example, if the file name is SHOW_QUEUE_ CHARACTERISTICS, and if you requested DIRECTORY to display both file name and size in each column, the display for that file would be SHOW_QUEUE_CHARACT | 120.

/CREATED (default)

Modifies the time value specified with the /BEFORE or the /SINCE qualifier. The /CREATED qualifier selects files based on their dates of creation. This qualifier is incompatible with the /BACKUP, /EXPIRED, and /MODIFIED qualifiers, which also allow you to select files according to time attributes. If you specify none of these four time qualifiers, the default is the /CREATED qualifier.

/DATE[=option]
/NODATE (default)

Includes the backup, creation, expiration, or modification date for each specified file; the default is the /NODATE qualifier. If you use the /DATE qualifier without an option, the creation date is provided. Possible options are as follows:

ALL	Specifies creation, expiration, backup, and last modification dates.
BACKUP	Specifies the last backup date.
CREATED	Specifies the creation date.
EXPIRED	Specifies the expiration date.
MODIFIED	Specifies the last modification date.

/EXCLUDE=(filespec[,...])

Excludes the specified files from the DIRECTORY command. You can include a directory but not a device in the file specification. Wildcard characters (* and %) are allowed in the file specification. However, you cannot use relative version numbers to exclude a specific version. If you specify only one file, you can omit the parentheses.

/EXPIRED

Modifies the time value specified with the /BEFORE or the /SINCE qualifier. The /EXPIRED qualifier selects files according to their expiration dates. (The expiration date is set with the SET FILE/EXPIRATION_DATE command.) The /EXPIRED qualifier is incompatible with the /BACKUP, /CREATED, and /MODIFIED qualifiers, which also allow you to select files according to time attributes. If you specify none of these four time qualifiers, the default is the /CREATED qualifier.

/FILE_ID

Controls whether the file identification number (FID) is displayed. By default, the FID is not displayed unless the /FULL qualifier is specified.

/FULL

Displays the following information for each file:

File name
File type
Version number
Number of blocks used
Number of blocks allocated
Date of creation
Date last modified and revision number
Date of expiration
Date of last backup
File owner's user identification code (UIC)
File protection
File identification number (FID)
File organization
Journaling information
Other file attributes
Record attributes
Record format
Access control list (ACL)
Value of the stored semantics tag (where applicable)

/GRAND_TOTAL

Displays only the totals for all files and directories that have been specified. Suppresses both the per-directory total and individual file information. (See the /TRAILING qualifier for information on displaying directory totals.)

/HEADING
/NOHEADING

Controls whether heading lines consisting of a device description and directory specification are printed. The default output format provides this heading. When the /NOHEADING qualifier is specified, the display is in single-column format and the device and directory information appears with each file name. The /NOHEADING qualifier overrides the /COLUMNS qualifier.

The combination of the /NOHEADING and /NOTRAILING qualifiers is useful in command procedures where you want to create a list of complete file specifications for later operations.

/MODIFIED

Modifies the time value specified with the /BEFORE or the /SINCE qualifier. The /MODIFIED qualifier selects files according to the dates on which they were last modified. This qualifier is incompatible with the /BACKUP, /CREATED, and /EXPIRED qualifiers, which also allow you to select files according to time attributes. If you specify none of these four time modifiers, the default is the /CREATED qualifier.

/OUTPUT[=filespec]
/NOOUTPUT

Controls where the output of the command is sent. By default, the display is written to the current SYS$OUTPUT device. No wildcard characters are allowed.

If you enter the /OUTPUT qualifier with a partial file specification (for example, /OUTPUT=[JONES]), DIRECTORY is the default file name and LIS the default file type. If you enter the /NOOUTPUT qualifier, output is suppressed.

If the output will be written to a file in the same directory, the output file name will appear in the directory listing.

/OWNER
/NOOWNER (default)

Controls whether the file owner's user identification code (UIC) is listed.

The default size of the owner field is 20 characters. If the file owner's UIC exceeds the length of the owner field, the information will be truncated. The size of this field can be altered by specifying /WIDTH=OWNER, along with a value for the owner field. For more information, see the description of the /WIDTH qualifier.

/PRINTER

Puts the display in a file and queues the file to SYS$PRINT for printing under the name given by the /OUTPUT qualifier. If you do not specify the /OUTPUT qualifier, output is directed to a temporary file named DIRECTORY.LIS, which is queued for printing and then is deleted.

/PROTECTION
/NOPROTECTION (default)

Controls whether the file protection for each file is listed.

/SECURITY

Controls whether information about file security is displayed; using the /SECURITY qualifier is equivalent to using the /ACL, /OWNER, and /PROTECTION qualifiers together.

/SELECT=(keyword[,...])

Allows you to select files for display according to size. Choose one of the following keywords:

SIZE=MAXIMUM=n	Displays files that have fewer blocks than the value of n, which defaults to 1,073,741,823. Use with MINIMUM=n to specify a size range for files to be displayed.
SIZE=MINIMUM=n	Displays files that have blocks equal to or greater than the value of n, which defaults to 0. Use with MAXIMUM=n to specify a size range for files to be displayed.
SIZE=(MAXIMUM=n,MINIMUM=m)	Displays files whose block size falls within the specified MAXIMUM and MINIMUM range.

By default, file selection is based on other criteria.

/SINCE[=time]

Selects only those files dated after the specified time. You can specify time as an absolute time, as a combination of absolute and delta times, or as one of the following keywords: TODAY (default), TOMORROW, or YESTERDAY. Specify one of the following qualifiers with the /SINCE qualifier to indicate the time attribute to be used as the basis for selection: /BACKUP, /CREATED (default), /EXPIRED, or /MODIFIED.

For complete information on specifying time values, see the *VMS DCL Concepts Manual*.

/SIZE[=option]
/NOSIZE (default)

Displays the size in blocks of each file. If you omit the option parameter, the default lists the file size in blocks used (USED). Specify one of the following options:

ALL   Lists the file size both in blocks allocated and blocks used.

ALLOCATION   Lists the file size in blocks allocated.

USED   Lists the file size in blocks used.

The size of this field can be altered by supplying the size value of the /WIDTH qualifier.

/TOTAL

Displays only the directory name and total number of files.

By default, the output format is determined by the /BRIEF qualifier, which gives this total but also lists all the file names, file types, and their version numbers.

/TRAILING
/NOTRAILING

Controls whether trailing lines that provide the following summary information are displayed:

* Number of files listed

* Total number of blocks used per directory

* Total number of blocks allocated

* Total number of directories and total blocks used or allocated in all directories (only if more than one directory is listed)

By default, the output format includes most of this summary information. The /SIZE and /FULL qualifiers determine more precisely what summary information is included.

When used alone, the /TRAILING qualifier lists the number of files in the directory. When used with the /SIZE qualifier, the /TRAILING qualifier lists the number of files and the number of blocks (displayed according to the option of the /SIZE qualifier, FULL or ALLOCATION). When used with the /FULL qualifier, the /TRAILING qualifier lists the number of files as well as the number of blocks used and allocated. If more than one directory is listed, the summary includes the total number of directories, the total number of blocks used, and the total number of blocks allocated.

/VERSIONS=n

Specifies the number of versions of a file to be listed. The default is all versions of each file. A value less than 1 is not allowed.

/WIDTH=(keyword[,...])

Formats the width of the display. If you specify only one keyword, you can omit the parentheses. Possible keywords are as follows:

DISPLAY=n   Specifies the total width of the display as an integer in the range 1 to 256 and defaults to zero (setting the display width to the terminal width). If the total width of the display exceeds the terminal width, the information will be truncated.

FILENAME=n   Specifies the width of the file name field; defaults to 19 characters. If you request another piece of information to be displayed along with the file name in each column, file names that exceed the n parameter cause the line to wrap after the file name field. (See the /COLUMNS qualifier.)

OWNER=n   Specifies the width of the owner field; defaults to 20 characters. If the owner's user identification code (UIC) exceeds the length of the owner field, the information will be truncated.

SIZE=n   Specifies the width of the size field; defaults to 6 characters. If the file size exceeds the length of the size field, the information will be truncated.

EXAMPLES

1   `$ DIRECTORY AVERAGE.*`

```
Directory DISK$DOCUMENT:[MALCOLM]

AVERAGE.EXE;6      AVERAGE.FOR;6      AVERAGE.LIS;4      AVERAGE.OBJ;12

Total of 4 files.
```

In this example, the DIRECTORY command lists all files with the file name AVERAGE and any file type.

2   `$ DIRECTORY/SIZE=USED/DATE=CREATED/VERSIONS=1/PROTECTION   AVERAGE`

```
Directory DISK$DOCUMENT:[MALCOLM]

AVERAGE.EXE;6        6        19-APR-1990 15:43:02.10  (RWED,RWED,RWED,RE)
AVERAGE.FOR;6        2        19-APR-1990 10:29:53.37  (RWED,RWED,RWED,RE)
AVERAGE.LIS;4        5        19-APR-1990 16:27:27.19  (RWED,RWED,RWED,RE)
AVERAGE.OBJ;6        2        19-APR-1990 16:27:44.23  (RWED,RWED,RWED,RE)

Total of 4 files, 15 blocks.
```

In this example, the DIRECTORY command lists the number of blocks used, the creation date, and the file protection code for the highest version number of all files named AVERAGE in the current directory.

3 $ DIRECTORY/FULL [JONES.ITALIA]PROJECTIONS.LIS

Directory WORK:[JONES.ITALIA]

```
PROJECTIONS.LIS;1          File ID:   (7449,36222,2)
Size:          21/21       Owner:     [DOC,JONES]
Created:    5-MAY-1988 15:49:03.11
Revised:    5-MAY-1988 15:49:49.39 (2)
Expires:    <None specified>
Backup:     <No backup recorded>
File organization:  Sequential
File attributes:    Allocation: 21, Extend: 0, Global buffer count: 0,
                    No version limit
Record format:      Variable length, maximum 80 bytes
Record attributes:  Carriage return carriage control
RMS attributes:     None
Journaling enabled: None
File protection:    System:RWED, Owner:RWED, Group:RE, World:
Access Cntrl List:  None

Total of 1 file, 21/21 blocks.
```

The DIRECTORY command in this example shows the date/time format using the default language, English, and the default VMS format. You can also select other languages and formats that have been defined on your systems with international date/time formatting routines available in the run-time library. See the *VMS RTL Library (LIB$) Manual*.

4 $ DIRECTORY/VERSIONS=1/COLUMNS=1 AVERAGE.*

The DIRECTORY command in this example lists only the highest version of each file named AVERAGE in the current default directory. The format is brief and restricted to one column. Heading and trailing lines are provided.

5 $ DIRECTORY BLOCK%%%

The DIRECTORY command in this example locates all versions and types of files in the default device and directory whose names begin with the letters BLOCK and end with any three additional characters. The default output format is brief, four columns, with heading and trailing lines.

6 $ DIRECTORY/EXCLUDE=(AVER.DAT;*,AVER.EXE;*) [*...]AVER

The DIRECTORY command in this example lists and totals all versions and types of files named AVER in all directories and subdirectories on the default disk, except any files named AVER.DAT and AVER.EXE.

7 $ DIRECTORY/SIZE=ALL FRESNO::DISK1:[TAYLOR]*.COM

The DIRECTORY command in this example lists all versions of all files with the file type COM in the directory TAYLOR on node FRESNO and device DISK1. The listing includes the file size both in blocks used and in blocks allocated for each file.

```
8    $ DIRECTORY-
    _$ /MODIFIED/SINCE=19-APR-1990:01:30/SIZE=ALL/OWNER-
    _$ /PROTECTION/OUTPUT=UPDATE/PRINTER [A*]
```

The DIRECTORY command in this example locates all files that have been modified since 1:30 A.M. on April 19, 1990, and that reside on the default disk in all directories whose names begin with the letter A. It formats the output to include all versions, the size used and size allocated, the date last modified, the owner, and the protection codes. The output is directed to a file named UPDATE.LIS, which is queued automatically to the default printer queue and then is deleted.

DISCONNECT

Breaks the connection between a physical terminal and a virtual terminal. After the physical terminal is disconnected, both the virtual terminal and the process using it remain on the system.

Requires that your physical terminal is connected to a virtual terminal.

FORMAT

DISCONNECT

PARAMETERS

None.

DESCRIPTION

Use the DISCONNECT command to disconnect a physical terminal from a virtual terminal and its associated process. The virtual terminal and the process remain on the system, so you can use the CONNECT command to reconnect to the process later. (For more information about virtual terminals and how to connect to them, see the description of the CONNECT command.) To terminate a process connected to a virtual terminal, use the LOGOUT command.

After you are disconnected from a virtual terminal, you can use the physical terminal to log in again.

You can use the DISCONNECT command only if your physical terminal is connected to a virtual terminal.

QUALIFIER

/CONTINUE
/NOCONTINUE (default)
Controls whether the CONTINUE command is executed in the current process just before connecting to another process. This procedure permits an interrupted image to continue processing after the disconnection until the process needs terminal input or attempts to write to the terminal. At that point, the process waits until the physical terminal is reconnected to the virtual terminal.

EXAMPLES

1   `$ DISCONNECT`

This command disconnects a physical terminal from a virtual terminal, but does not log the process out. Now you can use the physical terminal to log in again.

2 $ RUN PAYROLL
      `Ctrl/Y`
      $ DISCONNECT/CONTINUE

In this example, the RUN command is entered from a physical terminal that is connected to a virtual terminal. After the image PAYROLL.EXE is interrupted, the DISCONNECT command disconnects the physical and the virtual terminals without logging out the process. The /CONTINUE qualifier allows the image PAYROLL.EXE to continue to execute until the process needs terminal input or attempts to write to the terminal. At that point, the process waits until the physical terminal is reconnected to the virtual terminal. However, you can use the physical terminal to log in again and perform other work.

DISMOUNT

Closes a mounted disk or magnetic tape volume for further processing and cancels the logical name associated with the device.

Requires the GRPNAM (group logical name) and SYSNAM (system logical name) privileges to dismount group and system volumes.

FORMAT   **DISMOUNT**   *device-name[:]*

PARAMETER   ***device-name[:]***
Name of the device containing the volume—either a logical name or a physical name. If a physical name is specified, the controller defaults to A and the unit defaults to 0.

If the volume currently mounted on the device is a member of a disk or tape volume set, all volumes in the set are dismounted, unless the /UNIT qualifier is specified.

DESCRIPTION   The DISMOUNT command (which invokes the $DISMOU system service) checks for conditions that prevent a Files–11 volume from dismounting. The conditions fall into the following four categories:

- Installed swap and page files

- Installed images

- Devices spooled to the volume

- Open user files (any files not falling into one of the first three categories)

If the DISMOUNT command does not find any of these conditions, it performs the following operations:

- Removes the volume from the user's list of mounted volumes, deletes the logical name (if any) associated with the volume, and decrements the mount count.

- If the mount count equals zero after being decremented, the DISMOUNT command marks the volume for dismounting.

  As soon as the volume is idle, that is, after the DISMOUNT command has determined that no user has any open files on the volume, the DISMOUNT command marks a Files–11 volume for dismounting, and dismounts the volume soon.

- If the mount count does not equal zero after being decremented, the DISMOUNT command does not mark the volume for dismount (because the volume must have been mounted shared). In this case, the total effect for the issuing process is that the process is denied access to the volume and the logical name is deleted.

- After a volume is dismounted, nonpaged pool is returned to the system. Paged pool is also returned if the volume was mounted using the /GROUP or /SYSTEM qualifiers.

If the DISMOUNT command does find open files or any other condition that prevents the volume from dismounting, it does *not* mark the volume for dismounting. Instead, the DISMOUNT command displays a message indicating that the volume cannot be dismounted, followed by messages indicating the conditions that exist and the number of instances of each condition.

The /OVERRIDE=CHECKS qualifier allows a volume to be marked for dismounting despite open files or other conditions. For example, marking a volume for dismounting prevents any new files from being opened. Also, when a volume is marked for dismounting, file-system caches are flushed. This activity is especially important when the system is shutting down and the file-system caches must be written to the disk.

If a volume is part of a Files–11 volume set and the /UNIT qualifier is not specified, the entire volume set will be dismounted.

If the volume was mounted with the /SHARE qualifier, it is not actually dismounted until all users who mounted it dismount it or log out. However, the DISMOUNT command deletes the logical name associated with the device.

If the device was allocated with an ALLOCATE command, it remains allocated after the volume is dismounted with the DISMOUNT command. If the device was implicitly allocated by the MOUNT command, the DISMOUNT command deallocates it.

If the volume was mounted with the /GROUP or the /SYSTEM qualifier, it is dismounted even if other users are currently accessing it. The GRPNAM and SYSNAM user privileges are required to dismount group and system volumes, respectively.

QUALIFIERS

/ABORT

Requires volume ownership or the user privilege VOLPRO (volume protection) to use this qualifier with a volume that was mounted with neither the /GROUP nor the /SYSTEM qualifier. Additionally requires the user privilege SHARE if the volume is mounted privately by a process other than the process issuing the DISMOUNT command.

Specifies that the volume is to be dismounted, regardless of who mounted it. The primary purpose of the /ABORT qualifier is to terminate mount verification. The DISMOUNT/ABORT command also cancels any outstanding I/O requests. If the volume was mounted with the /SHARE qualifier, the /ABORT qualifier causes the volume to be dismounted for all of the users who mounted it.

/CLUSTER

Dismounts a volume clusterwide. If you specify DISMOUNT/CLUSTER, the DISMOUNT command checks for open files or other conditions that will prevent a Files–11 volume on the local node from dismounting. If the DISMOUNT command does not find any open files or other conditions, it

checks for conditions on all other nodes in the cluster. If the DISMOUNT command finds one of the conditions on any node, it displays an error message identifying the device and the nodes on which the error occurred, followed by an error message indicating open files or other conditions on the volume.

After the DISMOUNT command successfully dismounts the volume on the local node, it dismounts the volume on every other node in the existing VAXcluster environment. If the system is not a member of a cluster, the /CLUSTER qualifier has no effect.

/OVERRIDE=CHECKS

Marks a Files–11 volume for dismounting even if files are open on the volume. If you specify DISMOUNT/OVERRIDE=CHECKS, the DISMOUNT command displays messages indicating any open files or other conditions that prevent dismounting, immediately followed by a message indicating that the volume has been marked for dismounting.

A substantial amount of time can pass between the time you enter the DISMOUNT/OVERRIDE=CHECKS command and the completion of the dismount operation. Always wait for the dismount to complete before you remove the volume. (To verify that the dismount has completed, enter the SHOW DEVICES command.) Note that the final phase of volume dismounting occurs in the file system, and all open files on the volume must be closed before the actual dismount can be done. Note also that the file system cannot dismount a volume while any known file lists associated with it contain entries.

/UNIT

Dismounts only one volume of a volume set on the specified device. By default, all volumes in a set are dismounted.

Note: **Avoid dismounting the root volume of a volume set, because it contains the master file directory (MFD). It may be impossible to access files on a volume set if the MFD is not accessible.**

/UNLOAD
/NOUNLOAD

Determines whether the device on which the volume is mounted is physically unloaded. If you specify the DISMOUNT command without the /UNLOAD or the /NOUNLOAD qualifier, the qualifier that you specified with the MOUNT command (either /UNLOAD or /NOUNLOAD) determines whether the volume is unloaded physically.

EXAMPLES

1 $ MOUNT MTA0: PAYVOL TAPE
     .
     .
     .
    $ DISMOUNT TAPE

The MOUNT command in this example mounts the tape whose volume identification is PAYVOL on the device MTA0: and assigns the logical name TAPE to the device. By default, the volume is not shareable. The

DISMOUNT command releases access to the volume, deallocates the device, and deletes the logical name TAPE.

2 $ MOUNT/SHARE DBA3: DOC_FILES
   .
   .
   .
   $ DISMOUNT DBA3:

The MOUNT command in this example mounts the volume labeled DOC_FILES on the device DBA3. Other users can enter MOUNT commands to access the device. The DISMOUNT command shown in this example deaccesses the device for the process issuing the command. If other users still have access to the volume, the volume remains mounted for their process or processes.

3 $ DISMOUNT/NOUNLOAD DMA2:

The DISMOUNT command in this example dismounts the volume; the /NOUNLOAD qualifier requests that the volume remain in a ready state.

4 $ MOUNT/BIND=PAYROLL DMA1:,DMA2: PAYROLL01,PAYROLL02
   .
   .
   .
   $ DISMOUNT/UNIT DMA2:

The MOUNT command in this example mounts PAYROLL, a two-volume set. The DISMOUNT command dismounts only PAYROLL02, leaving PAYROLL01 accessible. Note that because the master file directory (MFD) for the volume set is on the root volume, you should not dismount the root volume (in this case, PAYROLL01) of the volume set.

5 $ DISMOUNT 10DJA100
   %DISM-W-CANNOTDMT, 10DJA100: cannot be dismounted
   %DISM-W-INSWPGFIL, 4 swap or page files installed on volume
   %DISM-W-SPOOLEDEV, 3 devices spooled to volume
   %DISM-W-INSTIMAGE, 7 images installed on volume
   %DISM-W-USERFILES, 6 user files open on volume

The DISMOUNT command in this example displays the open files and other conditions that prevent device 10DJA100 from dismounting.

6 $ DISMOUNT/CLUSTER 10DJA100
   %DISM-W-RMTDMTFAIL, 10DJA100: failed to dismount on node SALT
   %DISM-W-FILESOPEN, volume has files open on remote node
   %DISM-W-RMTDMTFAIL, 10DJA100: failed to dismount on node PEPPER
   %DISM-W-FILESOPEN, volume has files open on remote node
   %DISM-W-CANNOTDMT, 10DJA100: cannot be dismounted

The DISMOUNT command in this example displays messages identifying device 10DJA100 and nodes SALT and PEPPER on which errors occurred followed by messages indicating open files on the volume.

DUMP

Displays the contents of a file, a disk volume, or a magnetic tape volume in decimal, hexadecimal, or octal format, as well as the ASCII conversion.

FORMAT   **DUMP**   *filespec [,...]*

PARAMETER   ***filespec [,...]***
Specifies the file or name of the device being dumped.

If the specified device is not a disk, a tape, or a network device, or if the device is mounted with the /FOREIGN qualifier, the file specification must contain only the device name.

If the specified device is a network device, a disk device, or a tape device that is mounted without the /FOREIGN qualifier, the file specification can contain wildcards.

DESCRIPTION   By default, the DUMP command formats the output both in ASCII characters and in hexadecimal longwords. You can specify another format for the dump by using a radix qualifier (/OCTAL, /DECIMAL, or /HEXADECIMAL) or a length qualifier (/BYTE, /WORD, or /LONGWORD).

Dumping Files

If the input medium is a network device, a disk device, or a tape device that is mounted without the /FOREIGN qualifier, the DUMP command operates on files. You can dump files by either records or blocks. Wildcard specifications can be used to select a group of files for processing.

Dumping Volumes

If the input medium is not a disk or a tape device, or if it is mounted with the /FOREIGN qualifier, the DUMP command operates on the input device as a non-file-structured (NFS) medium. Disk devices are dumped by 512-byte logical blocks. Other devices are dumped by physical blocks. No repositioning of the input medium occurs; therefore, consecutive blocks on a tape can be dumped by a single DUMP command.

If you have LOG_IO (logical I/O) privilege, you can dump random blocks on a Files–11 volume. For example, by using the /BLOCKS qualifier, you could dump block 100 on the system disk.

Reading Dumps

The ASCII representation is read left to right. The hexadecimal, decimal, and octal representations are read right to left.

Specifying Numeric Qualifier Values

The numeric values for the /BLOCKS, /RECORDS, and /NUMBER qualifiers can be specified either as decimal numbers or with a leading %X, %O, or %D to signify hexadecimal, octal, or decimal numbers respectively. For example, the following are all valid ways to specify decimal value 24:

```
24
%X18
%O30
%D24
```

QUALIFIERS	**/ALLOCATED**

/ALLOCATED

Includes in the dump all blocks allocated to the file. (By default, the dump does not include blocks following the end-of-file [EOF].)

You can specify the /ALLOCATED qualifier if the input is a disk that is mounted without the /FOREIGN qualifier. The /ALLOCATED and /RECORDS qualifiers are mutually exclusive.

/BLOCKS[=(option[,...])]

Dumps the specified blocks one block at a time, which is the default method for all devices except network devices.

Block numbers are specified as integers relative to the beginning of the file. Typically, blocks are numbered beginning with 1. If a disk device is mounted using the /FOREIGN qualifier, blocks are numbered beginning with zero. Select a range of blocks to be dumped by specifying one of the following options:

START:*n*	Specifies the number of the first block to be dumped; the default is the first block.
END:*n*	Specifies the number of the last block to be dumped; the default is the last block or the end-of-file (EOF) block, depending on whether you have specified the /ALLOCATED qualifier.
COUNT:*n*	Specifies the number of files to be dumped. The COUNT option provides an alternative to the END option; you cannot specify both.

If you specify only one option, you can omit the parentheses.

The /BLOCKS and /RECORDS qualifiers are mutually exclusive.

Use the /BLOCKS qualifier to dump random blocks from Files–11 volumes. This procedure requires LOG-IO (logical I/O) privilege.

/BYTE

Formats the dump in bytes. The /BYTE, /LONGWORD, and /WORD qualifiers are mutually exclusive. The default format is composed of longwords.

/DECIMAL

Dumps the file in decimal radix. The /DECIMAL, /HEXADECIMAL (default), and /OCTAL qualifiers are mutually exclusive.

/FILE_HEADER

Dumps each data block that is a valid Files–11 header in Files–11 header format rather than in the selected radix and length formats.

/FORMATTED (default)
/NOFORMATTED

Dumps the file header in Files–11 format; the /NOFORMATTED qualifier dumps the file header in octal format. This qualifier is useful only when the /HEADER qualifier is specified.

/HEADER

Dumps the file header and access control list (ACL). To dump only the file header, and not the file contents, also specify /BLOCK=(COUNT:0). The /HEADER qualifier is invalid for devices mounted using the /FOREIGN qualifier.

Use the /FORMATTED qualifier to control the format of the display.

You can use the /FILE_HEADER qualifier with the /HEADER qualifier to have Files–11 file headers printed in an interpreted representation.

By default, the file header is not displayed.

/HEXADECIMAL (default)

Dumps the file in hexadecimal radix. The /DECIMAL, /HEXADECIMAL (default), and /OCTAL qualifiers are mutually exclusive.

/LONGWORD (default)

Formats the dump in longwords. The /BYTE, /LONGWORD, and /WORD qualifiers are mutually exclusive.

/NUMBER[=n]

Specifies how byte offsets are assigned to the lines of output. If you specify the /NUMBER qualifier, the byte offsets increase continuously through the dump, beginning with n; if you omit the /NUMBER qualifier, the first byte offset is zero. By default, the byte offset is reset to zero at the beginning of each block or record.

/OCTAL

Dumps the file in octal radix. The /DECIMAL, /HEXADECIMAL (default), and /OCTAL qualifiers are mutually exclusive.

/OUTPUT[=filespec]

Specifies the output file for the dump. If you do not specify a file specification, the default is the file name of the file being dumped and the file type DMP. If the /OUTPUT qualifier is not specified, the dump goes to SYS$OUTPUT. No wildcard characters are allowed. The /OUTPUT and /PRINTER qualifiers are mutually exclusive.

/PRINTER

Queues the dump to SYS$PRINT in a file named with the file name of the file being dumped and the file type DMP. If the /PRINTER qualifier is not specified, the dump goes to SYS$OUTPUT. No wildcard characters are allowed. The /OUTPUT and /PRINTER qualifiers are mutually exclusive.

/RECORDS[=(option[,...])]

Dumps the file a record at a time rather than a block at a time. (By default, input is dumped one block at a time for all devices except network devices.)

Blocks are numbered beginning with 1.

Select a range of blocks to be dumped by specifying one of the following options:

START:*n*   Specifies the number of the first record to be dumped; the default is the first record.

END:*n*   Specifies the number of the last record to be dumped; the default is the last record of the file.

COUNT:*n*   Specifies the number of records to be dumped. The COUNT option provides an alternative to the END option; you cannot specify both.

If you specify only one option, you can omit the parentheses.

If you specify the /RECORDS qualifier, you cannot specify the /ALLOCATED or the /BLOCKS qualifier.

/WORD
Formats the dump in words. The /BYTE, /LONGWORD, and /WORD qualifiers are mutually exclusive.

EXAMPLES

1
```
$ DUMP TEST.DAT
Dump of file DISK0:[NORMAN]TEST.DAT;1 on 19-APR-1990 15:43:26.08
File ID (3134,818,2)   End of file block 1 / Allocated 3
Virtual block number 1 (00000001), 512 (0200) bytes
 706D6173 20612073 69207369 68540033 3.This is a samp 000000
 73752065 62206F74 20656C69 6620656C le file to be us 000010
 61786520 504D5544 2061206E 69206465 ed in a DUMP exa 000020
 00000000 00000000 0000002E 656C706D mple............ 000030
 00000000 00000000 00000000 00000000 ................ 000040
 00000000 00000000 00000000 00000000 ................ 000050
 00000000 00000000 00000000 00000000 ................ 000060
                      .
                      .
                      .
 00000000 00000000 00000000 00000000 ................ 0001E0
 00000000 00000000 00000000 00000000 ................ 0001F0
```

The DUMP command displays the contents of TEST.DAT both in hexadecimal longword format and in ASCII beginning with the first block in the file.

DUMP

2 ```
$ DUMP TEST.DAT/OCTAL/BYTE
Dump of file DISK0:[NORMAN]TEST.DAT;1 on 19-APR-1990 15:45:33.58
File ID (74931,2,1)   End of file block 1 / Allocated 3
Virtual block number 1 (00000001), 512 (0200) bytes
 151 040 163 151 150 124 000 063 3.This i 000000
 160 155 141 163 040 141 040 163 s a samp 000010
 040 145 154 151 146 040 145 154 le file   000020
 163 165 040 145 142 040 157 164 to be us 000030
 040 141 040 156 151 040 144 145 ed in a   000040
 141 170 145 040 120 115 125 104 DUMP exa 000050
 377 377 000 056 145 154 160 155 mple.... 000060
 000 000 000 000 000 000 000 000 000070
 000 000 000 000 000 000 000 000 000100
 000 000 000 000 000 000 000 000 000110
                 .
                 .
                 .
 000 000 000 000 000 000 000 000 000760
 000 000 000 000 000 000 000 000 000770
```

> The DUMP command displays the image of the file TEST.DAT, formatted both in octal bytes and in ASCII characters beginning with the first block.

**3**  ```
$ DUMP NODE3::DISK2:[STATISTICS]RUN1.DAT
```

> This command line dumps the file RUN1.DAT that is located at remote node NODE3. The default DUMP format will be used.

EDIT/ACL

Invokes the Access Control List (ACL) Editor, which creates or modifies an access control list for a specified object. The /ACL qualifier is required. For a complete description of the ACL Editor, see the *VMS Access Control List Editor Manual*.

FORMAT **EDIT/ACL** *object-spec*

EDIT/EDT

Invokes the DIGITAL Standard Editor (EDT) interactive text editor. The /EDT qualifier is not required, because EDT is the VMS default editor.

FORMAT

EDIT *filespec*

PARAMETER

filespec

Specifies the file to be created or edited using EDT. If the file does not exist, it is created by EDT.

EDT does not provide a default file type when creating files; if you do not include a file type, it is null. The file must be a disk file on a Files–11 formatted volume.

No wildcard characters are allowed in the file specification.

DESCRIPTION

EDT creates or edits text files. You can use EDT to enter or edit text in three modes: keypad, line, or nokeypad. Keypad editing, which is screen-oriented, is available on VT300-series, VT200-series, VT100, and VT52 terminals. A screen-oriented editor allows you to see several lines of text at once and move the cursor throughout the text in any direction. Line editing operates on all terminals. In fact, if you have a terminal other than a VT300-series, VT200-series, VT100, or VT52, line editing is the only way you can use EDT. You might prefer line editing if you are accustomed to editing by numbered lines. Nokeypad mode is a command-oriented screen editor available on VT300-series, VT200-series, VT100, and VT52 terminals. You can use line mode and nokeypad mode to redefine keys for use in keypad mode.

When you invoke EDT, you are in line mode by default. If you are editing an existing file, EDT displays the line number and text for the first line of the file. If you are creating a new file, EDT displays the following message:

```
Input file does not exist
[EOB]
```

In either case, EDT then displays the line mode prompt, which is the asterisk (*).

For complete details on the EDT editor, see the *VAX EDT Reference Manual*.

QUALIFIERS

/COMMAND[=filespec]
/NOCOMMAND

Determines whether or not EDT uses a startup command file. The /COMMAND file qualifier should be followed by an equal sign (=) and the specification of the command file. The default file type for command files is EDT. No wildcard characters are allowed in the file specification.

The following command line invokes EDT to edit a file named MEMO.DAT and specifies that EDT use a startup command file named XEDTINI.EDT:

```
$ EDIT/COMMAND=XEDTINI.EDT MEMO.DAT
```

If you do not include the /COMMAND=command file qualifier, EDT looks for the EDTSYS logical name assignment. If EDTSYS is not defined, EDT processes the systemwide startup command file SYS$LIBRARY:EDTSYS.EDT. If this file does not exist, EDT looks for the EDTINI logical name assignment. If EDTINI is not defined, EDT looks for the file named EDTINI.EDT in your default directory. If none of these files exists, EDT begins your editing session in the default state.

To prevent EDT from processing either the systemwide startup command file or the EDTINI.EDT file in your default directory, use the /NOCOMMAND qualifier as follows:

```
$ EDIT/NOCOMMAND MEMO.DAT
```

/CREATE (default)
/NOCREATE

Controls whether EDT creates a new file when the specified input file is not found.

Normally, EDT creates a new file to match the input file specification if it cannot find the requested file name in the specified directory. When you use the /NOCREATE qualifier in the EDT command line and type a specification for a file that does not exist, EDT displays an error message and returns to the DCL command level as follows:

```
$ EDIT/NOCREATE NEWFILE.DAT
Input file does not exist
$
```

/JOURNAL[=journal-file]
/NOJOURNAL

Determines whether EDT keeps a journal during your editing session. A journal contains a record of the keystrokes you enter during an editing session. The default file name for the journal is the same as the input file name. The default file type is JOU. The /JOURNAL qualifier enables you to use a different file specification for the journal.

The following command line invokes EDT to edit a file named MEMO.DAT and specifies the name SAVE.JOU for the journal:

```
$ EDIT/JOURNAL=SAVE MEMO.DAT
```

If you are editing a file from another directory and want the journal to be located in that directory, you must use the /JOURNAL qualifier with a file specification that includes the directory name. Otherwise, EDT creates the journal in the default directory.

The directory that is to contain the journal should not be write-protected.

To prevent EDT from keeping a record of your editing session, use the /NOJOURNAL qualifier in the EDT command line as follows:

```
$ EDIT/NOJOURNAL MEMO.DAT
```

Once you have created a journal, enter the EDT/RECOVER command to execute the commands in the journal. No wildcard characters are allowed in the file specification.

/OUTPUT=output-file
/NOOUTPUT

Determines whether EDT creates an output file at the end of your editing session. The default file specification for both the input file and the output file is the same. Use the /OUTPUT qualifier to give the output file a different file specification from the input file.

The following command line invokes EDT to edit a file named MEMO.DAT and gives the resulting output file the name OUTMEM.DAT:

```
$ EDIT/OUTPUT=OUTMEM.DAT MEMO.DAT
```

You can include directory information as part of your output file specification to send output to another directory as follows:

```
$ EDIT/OUTPUT=[BARRETT.MAIL]MEMO.DAT MEMO.DAT
```

The /NOOUTPUT qualifier suppresses the creation of an output file, but not the creation of a journal. If you decide that you do not want an output file, you can use the /NOOUTPUT qualifier as follows:

```
$ EDIT/NOOUTPUT MEMO.DAT
```

A system interruption does not prevent you from re-creating your editing session because a journal is still being maintained. To save your editing session, even when you specify /NOOUTPUT, use the line mode command WRITE to put the text in an external file before you end the session.

No wildcard characters are allowed in the file specification.

/READ_ONLY
/NOREAD_ONLY (default)

Determines whether EDT keeps a journal and creates an output file. With the /NOREAD_ONLY qualifier, EDT maintains the journal and creates an output file when it processes the line mode command EXIT. Using the /READ_ONLY qualifier has the same effect as specifying both the /NOJOURNAL and /NOOUTPUT qualifiers.

The following command line invokes EDT to edit a file named CALENDAR.DAT, but does not create a journal or an output file:

```
$ EDIT/READ_ONLY CALENDAR.DAT
```

Use the /READ_ONLY qualifier when you are searching a file and do not intend to make any changes to it. To modify the file, use the line mode command WRITE to save your changes. Remember, however, that you have no journal.

/RECOVER
/NORECOVER (default)

Determines whether EDT reads a journal at the start of the editing session.

When you use the /RECOVER qualifier, EDT reads the appropriate journal and processes whatever commands it contains. The appropriate syntax is as follows:

```
$ EDIT/RECOVER MEMO.DAT
```

If the journal file type is not JOU or the file name is not the same as the input file name, you must include both the /JOURNAL qualifier and the /RECOVER qualifier as follows:

```
$ EDIT/RECOVER/JOURNAL=SAVE.XXX MEMO.DAT
```

Because the /NORECOVER qualifier is the default for EDT, you do not need to specify it in a command line.

EXAMPLES

1
```
$ EDIT/OUTPUT=NEWFILE.TXT OLDFILE.TXT
    1       This is the first line of the file OLDFILE.TXT.
*
```

This EDIT command invokes EDT to edit the file OLDFILE.TXT. EDT looks for the EDTSYS logical name assignment. If EDTSYS is not defined, EDT processes the systemwide startup command file SYS$LIBRARY:EDTSYS.EDT. If this file does not exist, EDT looks for the EDTINI logical name assignment. If EDTINI is not defined, EDT looks for the file named EDTINI.EDT in your default directory. If none of these files exists, EDT begins your editing session in the default state. When the session ends, the edited file has the name NEWFILE.TXT.

2
```
$ EDIT/RECOVER OLDFILE.TXT
```

This EDIT command invokes EDT to recover from an abnormal exit during a previous editing session. EDT opens the file OLDFILE.TXT, and then processes the journal OLDFILE.JOU. Once the journal has been processed, the user can resume interactive editing.

EDIT/FDL

Invokes the VMS File Definition Language (FDL) Editor, which creates and modifies FDL files. The /FDL qualifier is required. For a complete description of the File Definition Language Facility, see the *VMS File Definition Language Facility Manual.*

FORMAT   **EDIT/FDL**   *filespec*

EDIT/SUM

Invokes the SUMSLP Utility, a batch-oriented editor, to update a single input file with multiple files of edit commands.

For a complete description of the SUMSLP Utility, see the *VMS SUMSLP Utility Manual.*

FORMAT   **EDIT/SUM**   *input-file*

EDIT/TECO

Invokes the TECO interactive text editor. The /TECO qualifier is required.

FORMAT   **EDIT/TECO**   *[filespec]*
EDIT/TECO/EXECUTE=command-file   *[argument]*

PARAMETER   ***filespec***
Specifies the file to be created or edited using the TECO editor. If the file does not exist, it is created by TECO, unless you specify the /NOCREATE qualifier. No wildcard characters are allowed in the file specification.

If you specify the /MEMORY qualifier (default) without a file specification, TECO edits the file identified by the logical name TEC$MEMORY. If TEC$MEMORY has no equivalence string, or if the /NOMEMORY qualifier is specified, TECO starts in command mode and does not edit an existing file.

If you specify the /MEMORY qualifier and a file specification, the file specification is equated to the logical name TEC$MEMORY.

DESCRIPTION   The TECO editor creates or edits text files. For detailed information on the use of TECO, see the *PDP–11 TECO Editor Reference Manual*.

QUALIFIERS   ***/COMMAND[=filespec]***
/NOCOMMAND
Controls whether a startup command file is used. The /COMMAND file qualifier may be followed by an equal sign (=) and the specification of the command file. The default file type for command files is TEC.

The following command line invokes TECO to edit a file named MEMO.DAT and specifies that TECO use a startup command file named XTECOINI.TEC:

```
$ EDIT/TECO/COMMAND=XTECOINI.TEC MEMO.DAT
```

If you do not include the /COMMAND qualifier, or if you enter /COMMAND without specifying a command file, TECO looks for the TEC$INIT logical name assignment. If TEC$INIT is not defined, no startup commands are executed.

The logical name TEC$INIT can equate either to a string of TECO commands or to a dollar sign ($) followed by a file specification. If TEC$INIT translates to a string of TECO commands, the string is executed; if it translates to a dollar sign followed by a file specification, the contents of the file are executed as a TECO command string. For further information, see the *PDP–11 TECO Editor Reference Manual*.

To prevent TECO from using any startup command file, use the /NOCOMMAND qualifier as follows:

```
$ EDIT/TECO/NOCOMMAND MEMO.DAT
```

No wildcards are allowed in the file specification.

/CREATE (default)
/NOCREATE

Creates a new file when the specified input file cannot be found. If the /MEMORY qualifier is specified and no input file is specified, the file created is the one specified by the logical name TEC$MEMORY. Normally, TECO creates a new file to match the input file specification if it cannot find the requested file name in the specified directory. When you use the /NOCREATE qualifier in the TECO command line and type a specification for a file that does not exist, TECO displays an error message and returns you to the DCL command level. The /CREATE and /NOCREATE qualifiers are incompatible with the /EXECUTE qualifier.

/EXECUTE=command-file [argument]

Invokes TECO and executes the TECO macro found in the command file. The argument, if specified, appears in the text buffer when macro execution starts. Blanks or special characters must be enclosed in quotation marks (" "). For detailed information on the use of TECO macros, see the *PDP-11 TECO Editor Reference Manual*.

The /EXECUTE qualifier is incompatible with the /CREATE and /MEMORY qualifiers.

/MEMORY (default)
/NOMEMORY

Specifies that the last file you edited with TECO, identified by the logical name TEC$MEMORY, will be the file edited if you omit the file specification to the EDIT/TECO command.

/OUTPUT=output-file
/NOOUTPUT (default)

Controls how the output file is named at the end of your editing session. By default, the output file has the same name as the input file but is given the next higher available version number. Use the /OUTPUT qualifier to give the output file a file specification different from the input file.

The following command line invokes TECO to edit a file named MEMO.DAT and gives the resulting output file the name OUTMEM.DAT:

```
$ EDIT/TECO/OUTPUT=OUTMEM.DAT MEMO.DAT
```

You can include directory information as part of your output file specification to send output to another directory as follows:

```
$ EDIT/TECO/OUTPUT=[BARRRET.MAIL]MEMO.DAT MEMO.DAT
```

No wildcard characters are allowed in the file specification.

/READ_ONLY
/NOREAD_ONLY (default)

Controls whether an output file is created. By default, an output file is created; the /READ_ONLY qualifier suppresses the creation of the output file.

EXAMPLES

1 $ EDIT/TECO/OUTPUT=NEWFILE.TXT OLDFILE.TXT

This EDIT command invokes the TECO editor to edit the file OLDFILE.TXT. TECO looks for the TEC$INIT logical name assignment. If TEC$INIT is not defined, TECO begins the editing session without using a command file. When the session ends, the edited file has the name NEWFILE.TXT.

2 $ EDIT/TECO/EXECUTE=FIND_DUPS "TEMP, ARGS, BLANK"

In this example, the /EXECUTE qualifier causes the TECO macro contained in the file FIND_DUPS.TEC to be executed, with the argument string "TEMP, ARGS, BLANK" located in the text buffer.

EDIT/TPU

Invokes the VAX Text Processing Utility (VAXTPU). By default, this runs the Extensible VAX Editor (EVE). VAXTPU provides a structured programming language and other components for creating text editors and other applications. EVE is a general-purpose text editor that runs on DECwindows or character-cell terminals.

FORMAT

EDIT/TPU *[input-file]*

PARAMETER

input-file

Specifies the text file you want to create or edit. The file must be a disk file on a Files–11 formatted volume. There is no default file type—if you do not not specify a file type, the file type is null. The way the input file is processed depends on the VAXTPU application you are using. EVE handles the input file as follows:

- EVE uses the input file name and file type for the buffer name. If the input file exists, EVE copies it into the buffer in the main window. EVE displays a message telling you the number of lines in the file. For example, the following command edits a file named JABBER.TXT:

```
$ EDIT/TPU jabber.txt
24 lines read from DISK$1:[ALICE]JABBER.TXT;4
```

- If the input file does not exist—that is, if you are creating a new file— EVE creates an empty buffer, and displays a message that the file was not found. You can then begin typing and editing.

- If you do not specify an input file, EVE creates an empty buffer named Main. You can then begin typing and editing, or you can specify the file you want to edit or create by using the GET FILE, the OPEN, or the OPEN SELECTED command.

- EVE lets you use logical names and wildcards (* and %) to specify the file—for example, *.TXT. You can create and edit more than one file in an editing session, but you can specify only one input file on the command line. If more than one file matches your request, EVE displays a list of the matching files so you can choose the one you want. For more information, see the EVE online help topic called Choices Buffer.

- If the input file you specify is ambiguous, EVE delays applying the following qualifiers (or their defaults) until after you resolve the ambiguity:

  /[NO]MODIFY
  /[NO]OUTPUT
  /[NO]READ_ONLY
  /START_POSITION
  /[NO]WRITE

- If you use a search list to specify the file or use wildcards for the device or directory (such as [...]), EVE gets the first matching file in the search list or directory tree. If none of the files in the search list exists, EVE creates an empty buffer using the first file name in the search list (unless you used the /NOCREATE qualifier).

DESCRIPTION

The VAX Text Processing Utility (VAXTPU) provides a structured programming language, with an interpreter, compiler, and other components, for creating text editors and other applications. VAXTPU has a callable interface so you can call editing functions from a program written in BLISS, C, FORTRAN, or another language.

The Extensible VAX Editor (EVE) is a general-purpose text editor created with VAXTPU by default. You can use EVE on DECwindows or character-cell terminals (VT300, VT200, or VT100 series). EVE reads and writes standard ASCII text files. Because VAXTPU and EVE are expressly designed for the VMS operating system, they provide better performance than EDT or other editors.

Using EVE, you can do the following:

- Perform basic text editing and formatting operations, such as case change, copy, erase, fill, find, insert, paginate, and replace.

- Create or edit one or more files in an editing session.

- Use multiple buffers and windows to view and edit different files in the same editing session.

- Define keys for editing operations, including learn sequences (to bind several commands or keystrokes to a single key) and setting the EDT or WPS keypad.

- Select boxes or linear ranges for cut-and-paste or other edits.

- Use either VMS-style wildcards or ULTRIX-style wildcards to search for patterns of text.

- Execute DCL commands, such as DIRECTORY, from within the editor.

- Run DECspell to check selected text or an entire buffer.

- Spawn subprocesses or attach to other processes.

- Compile and execute VAXTPU procedures to extend EVE.

- Add or delete menu items for the DECwindows interface.

- Save procedures, menu definitions, key definitions, and other customizations for future sessions.

- Use initialization files to set editing preferences at startup or during an editing session.

- Recover your work in case of a system failure.

- Get comprehensive online help on EVE commands, keys, menu items, and other topics, including VAXTPU built-in procedures.

You may want to create a symbol for invoking EVE, by putting the following line in your LOGIN.COM file:

```
$ eve :== EDIT/TPU     ! my symbol to invoke EVE
```

To enter EVE commands, press the Do key or PF4, type the command, and press the Return key. To get a keypad diagram and help on defined keys, press the Help key (on VT100 terminals, press PF2). To exit from EVE, press F10 or Ctrl/Z.

Generally, you use EVE as an interactive, screen-oriented editor, although you can run EVE for batch jobs (using the /NODISPLAY qualifier and either the /COMMAND or the /INITIALIZATION qualifier). You can extend EVE to build your own VAXTPU application, effectively using EVE as a run-time library of VAXTPU procedures. The EVE source files are available in SYS$EXAMPLES. For a list of the EVE source files, use the following command:

```
$  DIRECTORY SYS$EXAMPLES:EVE$*.TPU
```

For information on VAXTPU programming, see the *VAX Text Processing Utility Manual*. For information about editing with EVE, see the *VMS EVE Reference Manual* or the *Guide to VMS Text Processing*.

QUALIFIERS

/COMMAND[=command-file] (default)
/NOCOMMAND

Determines the VAXTPU command file you want to use, if any. A command file contains procedures and executable statements to extend the editor. For example, you can use a command file to create additional EVE commands, define keys and menu items, or set attributes. You can also use a command file to set up a special text-processing environment for creating your own application or for batch editing.

You cannot use wildcards to specify the command file. You can specify one command file at a time. The default file type is TPU.

There are three ways to specify the command file you want to use:

*   Name the command file TPU$COMMAND.TPU.

    By default, VAXTPU looks for this command file in your current default directory. Thus, you can have a different TPU$COMMAND.TPU file for each directory.

*   Define the logical name TPU$COMMAND to specify the command file.

    This lets you use the same command file for all editing sessions, including when you invoke VAXTPU within MAIL, and lets you keep that command file in any convenient directory. Defining the logical name overrides the default search for the TPU$COMMAND.TPU file. You can put the definition in your LOGIN.COM file.

    For example, the following commands define TPU$COMMAND as a file named MYPROCS.TPU in your top-level, login directory, and then invoke VAXPTU using that command file:

```
$  DEFINE TPU$COMMAND sys$login:myprocs
$  EDIT/TPU
```

- Use the /COMMAND qualifier and specify the command file on the command line.

  This overrides any definition of the logical name TPU$COMMAND and overrides the default search for a TPU$COMMAND.TPU file. For example, the following command invokes VAXTPU, using a command file named MYPROCS.TPU in the current directory:

  ```
  $ EDIT/TPU/COMMAND=myprocs
  ```

If the command file you specify either with the /COMMAND qualifier or by defining the logical name TPU$COMMAND is not found, the editing session is aborted, returning you to the DCL (interactive) level.

If you do not want a command file executed—typically if you defined the logical name TPU$COMMAND or created a TPU$COMMAND.TPU file—use the /NOCOMMAND qualifier. Also, using the /NOCOMMAND qualifier makes startup faster because VAXTPU then does not search for a TPU$COMMAND.TPU file and does not have to compile and execute code startup.

At startup, VAXTPU compiles and executes a command file after loading a section file (if any) and before EVE executes an initialization file (if any). Thus, procedures, settings, and key definitions in a command file override those in the section file.

In EVE, you can use the SAVE ATTRIBUTES command to create or update a command file, saving most global settings ("attributes") and any menu definitions. For more information, see the *VMS EVE Reference Manual* or use the online help in EVE and read the topic called Attributes.

For more information about VAXTPU command files, see the *VAX Text Processing Utility Manual*, or use the online help in EVE and read the topic called Command Files.

/CREATE (default)
/NOCREATE

Determines whether a buffer is created if the input file is not found. The way this qualifier is processed depends on the VAXTPU application you are using.

For EVE, the default is the /CREATE qualifier. If the input file does not exist, EVE creates a buffer using the input file name and file type as the buffer name; or, if you do not specify an input file, EVE creates an empty buffer named Main.

Use the /NOCREATE qualifier to avoid invoking the editor in case you mistype the input file specification, or to edit only an existing file. Thus, if the input file is not found, the editing session is aborted, and you are returned to the DCL (interactive) level, as in the following example:

```
$ EDIT/TPU old.dat/NOCREATE
Input file does not exist: OLD.DAT;
$
```

/DEBUG[=debug-file]
/NODEBUG (default)

Determines whether to run a VAXTPU debugger. This is useful to test procedures for an application you are creating. VAXTPU compiles and

executes the debugger file before executing the initialization procedure TPU$INIT_PROCEDURE.

Using the /DEBUG qualifier (without specifying a debugger file) runs the default VAXTPU debugger, TPU$DEBUG.TPU, which provides commands to manipulate variables and to control program execution. To start editing the code in the file you are debugging, use the GO command. For more information about the debugger, read the comments in the TPU$DEBUG.TPU source file in SYS$SHARE, see the *VAX Text Processing Utility Manual*, or use the online help in EVE:

```
Command:  HELP TPU Debugger
```

VAXTPU assumes the debugger file is in SYS$SHARE. If your debugger file is stored elsewhere, specify the device (disk) and directory of the debugger file. You cannot use wildcards to specify the debugger file. You can use only one debugger file at a time. The default file type is TPU.

There are two ways to specify a debugger file of your own:

- Define the logical name TPU$DEBUG to specify the debugger file, and then use the command EDIT/TPU/DEBUG.

  Defining the logical name does *not* by itself run the debugger when you invoke VAXTPU. It only specifies which debugger file is run when you use the /DEBUG qualifier without specifying a debugger file. You can put the definition in your LOGIN.COM file.

- Use the /DEBUG qualifier and specify the debugger file on the command line.

  This overrides any definition of the TPU$DEBUG logical name. For example, the following command edits a file named MYPROCS.TPU, using a debugger file named MYDEBUG.TPU:

```
$  EDIT/TPU myprocs.tpu/DEBUG=mydebug
```

/DISPLAY[= $\left\{ \begin{array}{l} \textbf{\textit{CHARACTER\_CELL (default)}} \\ \textbf{\textit{DECWINDOWS}} \end{array} \right\}$ **]**
/NODISPLAY

Determines the type of screen display, if any. The /DISPLAY qualifier is the same as the /INTERFACE qualifier.

For example, the following command invokes VAXTPU with the DECwindows interface:

```
$  EDIT/TPU/INTERFACE=DECWINDOWS
```

Then, if DECwindows is available, VAXTPU displays the editing session in a separate window on your workstation screen and enables DECwindows features—for example, the EVE screen layout then includes a menu bar and scroll bars, and you can use MB1 (on the mouse) to move the cursor and select text. If DECwindows is not available, VAXTPU works as if on a character-cell terminal.

To specify your preferred display, you can define the logical name TPU$DISPLAY_MANAGER to be either CHARACTER_CELL or DECWINDOWS. However, as a general rule, do *not* define this logical name to be DECWINDOWS, because VAXTPU should be initialized only once using the DECwindows interface. Because of this restriction, utilities

that call VAXTPU multiple times (such as using the editor within MAIL) will fail.

Use the /NODISPLAY qualifier for batch jobs or when you are using an unsupported terminal. You typically use a VAXTPU command file or EVE initialization file for batch jobs, as in the following example, which uses a command file named BATCH.TPU:

```
$ EDIT/TPU/NODISPLAY/COMMAND=batch
```

This batch file should comprise a complete editing session, including the EXIT or the QUIT command. Note that some EVE commands cannot be used in batch, because they prompt for a key press or other interactive response.

/INITIALIZATION[=init-file] (default)
/NOINITIALIZATION

Determines the initialization file you want to use, if any. The way this qualifier is processed depends on the VAXTPU application you are using. An EVE initialization file contains a list of commands you want executed, typically to set margins, tab stops, and other attributes, or to define keys that you do not otherwise save in a section file.

You cannot use wildcards to specify the initialization file. You can specify only one initialization file at a time. The default file type is EVE.

There are three ways to specify the EVE initialization file you want to use:

- Name the initialization file EVE$INIT.EVE.

  By default, EVE first looks for this initialization file in your current default directory. If the file is not found there, EVE then looks for it in SYS$LOGIN (your top-level, login directory). Thus, you can have different initialization files for particular directories, and you can have a "standard" initialization file in SYS$LOGIN for editing in directories that do not have an EVE$INIT.EVE file.

- Define the logical name EVE$INIT to specify the initialization file.

  This lets you use the same initialization file for all editing sessions, including when you invoke EVE within MAIL, and lets you keep that initialization file in any convenient directory. Defining the logical name overrides the search for an EVE$INIT.EVE file. You can put the definition in your LOGIN.COM file.

  For example, the following commands define EVE$INIT to be a file named MYINIT.EVE in your top-level, login directory, and then invoke EVE using that initialization file:

```
$ DEFINE EVE$INIT sys$login:myinit
$ EDIT/TPU
```

- Use the /INITIALIZATION qualifier and specify an initialization file on the command line.

  This overrides any definition of the logical name EVE$INIT and overrides the default search for an EVE$INIT.EVE file. For example, the following command invokes EVE using an initialization file named MYINIT.EVE named in your current default directory:

```
$ EDIT/TPU/INITIALIZATION=myinit
```

If you do not want an initialization file executed—typically if you defined the logical name EVE$INIT or created an EVE$INIT.EVE file—use the /NOINITIALIZATION qualifier. Also, using /NOINITIALIZATION makes startup faster because EVE then does not search for an EVE$INIT.EVE file and does not parse commands at startup.

At startup, EVE executes an initialization file, if there is one, after VAXTPU loads the section file and compiles a command file (if any). Thus, settings and key definitions in an initialization file override those in a section file or command file. When you invoke EVE, commands in an initialization file for margins, tab stops, and other buffer settings apply to the main (or first) buffer and to an EVE system buffer named $DEFAULTS$. Buffers created during the session will have the same settings as $DEFAULTS$. For more information, use the online help in EVE and read the topic called Defaults.

If a command in an initialization file is incomplete—for example, if a command requires a file name, search string, or other parameter— EVE prompts you for the required information before going on. You can also execute an initialization file during an EVE session by using the @ command (at sign). This is useful when you want to set attributes or define keys for particular kinds of editing, or to execute a series of related commands.

An initialization file is somewhat slower than a section file or command file, depending on the number of commands to be executed. If you want to define several keys, you should save them in a section file. For more information about EVE initialization files, see the *VMS EVE Reference Manual*, or use the online help in EVE and read the topic called Initialization Files.

/INTERFACE[= { CHARACTER_CELL (default) / DECWINDOWS }]

Determines the type of interface or screen display. The /INTERFACE qualifier is the same as the /DISPLAY qualifier.

For example, the following command invokes VAXTPU with the DECwindows interface:

```
$  EDIT/TPU/INTERFACE=DECWINDOWS
```

Then, if DECwindows is available, VAXTPU displays the editing session in a separate window on your workstation screen and enables DECwindows features—for example, the EVE screen layout includes a menu bar and scroll bars, and you can use MB1 (on the mouse) to move the cursor and select text. If DECwindows is not available, VAXTPU works as if on a character-cell terminal. For information about using EVE on DECwindows, see the *VMS DECwindows Desktop Applications Guide*, or use the online help in EVE and read the topic called DECwindows Differences.

/JOURNAL[=journal-file] (default)
/NOJOURNAL

Determines the type of journaling, if any. Journaling records your edits so that if a system failure interrupts your editing session, you can recover your work. The way this qualifier is processed depends on the VAXTPU application you are using.

Note: **Journaling records information about the text you edit. Therefore, if you are editing confidential data, make sure the journals, as well as the text files, are secure.**

There are two types of journaling, as follows:

- **Buffer-change journaling** creates a journal for each text buffer. This is the EVE default. Buffer-change journaling works on DECwindows or character-cell terminals. The journal name derives from the name of the file or buffer being edited and the file type TPU$JOURNAL—for example:

Text Buffer	Buffer-Change Journal
MAIN	MAIN.TPU$JOURNAL
JABBER.TXT	JABBER_TXT.TPU$JOURNAL
GUMBO_RECIPE.RNO	GUMBO_RECIPE_RNO.TPU$JOURNAL
NEW TEST DATA	NEW_TEST_DATA.TPU$JOURNAL
* SCRATCH *	__SCRATCH__.TPU$JOURNAL

Buffer-change journals are created in the directory defined by the logical name TPU$JOURNAL. The default is SYS$SCRATCH, which is usually your top-level, login directory. Because buffer-change journals may be quite large—even larger than the files you edit— you may want to define the logical name TPU$JOURNAL as a disk and directory other than SYS$SCRATCH.

Some editing operations may be slower because of buffer-change journaling, depending on the extent or type of changes. For example, including a large file into the buffer or pasting a large amount of text from the DECwindows clipboard will be slower if you are using buffer-change journaling.

- **Keystroke journaling** creates a single journal for the editing session, regardless of the number of buffers you create. The journal records every keystroke in the editing session, whether text or commands. To enable keystroke journaling, use the /JOURNAL qualifier and specify the journal you want created. You cannot use wildcards to specify the journal. The default file type is TJL.

For example, the following command invokes VAXTPU, creating a keystroke journal named MYSESSION.TJL in your current default directory:

```
$  EDIT/TPU/JOURNAL=mysession
```

Keystroke journaling does *not* work on DECwindows and has other restrictions. Keystroke journaling is useful to reproduce a problem— for example, if you want to submit a software performance report (SPR)—or when you want to journal learn sequences and key definitions to be done during the editing session.

If you enable keystroke journaling, EVE also creates a buffer-change journal for each text buffer. This double journaling may slow performance, depending on the kind of edits you make. To disable buffer-change journaling during your editing session, use the TPU command SET NOJOURNALING.

Normally, journals are deleted when you exit or quit. If there is a system failure during your editing session, such as a communications break between your terminal and computer, the journals are saved. Your last few keystrokes or changes may be lost. For information about recovering your edits, see the description of the /RECOVER qualifier.

If you do not want any journaling, use the /NOJOURNAL qualifier, which disables both keystroke journaling and buffer-change journaling. This may make startup and some editing operations faster, but risks losing your work if there is system failure during the editing session. Typically, you use /NOJOURNAL with the /NOMODIFY, the /NOOUTPUT, the /READ_ONLY, or the /NOWRITE qualifier to view a file without making any changes. If you invoke EVE with /NOJOURNAL, you can enable buffer-change journaling during your editing session by using SET JOURNALING commands.

Note: **Although journaling and recovery are quite reliable, the safest way to protect your work against a system failure is to write out your edits frequently—particularly during all-day editing sessions.**

/MODIFY (default)
/NOMODIFY

Determines whether you can modify the main (or first) buffer. This does not affect other buffers you create during the editing session.

By default, VAXTPU allows the buffer to be modified—that is, you can edit text in the buffer, and exiting writes out the buffer to a file if the buffer has been modified (unless you used the /NOWRITE or /READ_ONLY qualifier). Use the /NOMODIFY qualifier to view a file without making any changes. You can then use cursor-movement commands, but cannot change the text.

For EVE, using the /READ_ONLY or /NOWRITE qualifier makes the buffer unmodifiable unless you also use the /MODIFY qualifier. For example, the following command edits a file named PRACTICE.TXT, making the buffer read-only and making it modifiable, so that you can practice editing or test procedures without writing out a file:

```
$ EDIT/TPU practice.txt/MODIFY/READ_ONLY
```

In EVE, the status line shows whether the buffer is unmodifiable. If the buffer is modifiable, the status line shows the mode (insert or overstrike). You can change the modification attribute of the buffer during your editing session by using the SET BUFFER command.

/OUTPUT[=output-file] (default)
/NOOUTPUT

Determines the output file specification, if any, for the main (or first) buffer. This does not affect other buffers you create during the editing session.

By default, the output file has the same specifications as the input file with a version number one higher than the highest existing version of the input file, or version 1 if you are creating a new file.

Use the /OUTPUT qualifier and specify a file if you want the output file to be written in a different directory or if you want the file to have a different name or file type. For example, the following command edits a file named ROUGH.LIS in your current, default directory and, on exiting, writes the output file to FINAL.TXT in your top-level, login directory:

```
$  EDIT/TPU rough.lis/OUTPUT=sys$login:final.txt
```

You cannot use wildcards to specify the output file. If you omit parts of the output file specification, such as the device (disk) or directory, VAXTPU uses the corresponding parts of the input file specification, if there is one.

Using /OUTPUT and specifying an output file modifies the buffer, so that even if you make no changes to the text, exiting writes out the buffer to a file.

For EVE, using the /NOOUTPUT qualifier makes the main (or first) buffer read-only (sometimes called write-locked), so that exiting does not write that buffer to a file. This is useful to view a file without making any changes. If you change your mind and want to write out the buffer before exiting, use the WRITE FILE, the SAVE FILE, or the SAVE FILE AS command. Also, you can change the read/write attribute of the buffer during your editing session by using the SET BUFFER command.

/READ_ONLY
/NOREAD_ONLY (default)

Determines whether exiting writes the main (or first) buffer to a file. This does not affect other buffers you create during the editing session.

The /READ_ONLY qualifier is the same as the /NOWRITE qualifier. For EVE, this makes the main (or first) buffer write-locked and also makes it unmodifiable, unless you use the /MODIFY qualifier. Use /READ_ONLY to view a file without making any edits. For example, the following command lets you view a file named STAFFMEMO.TXT so you can use cursor-movement commands but cannot change the text:

```
$  EDIT/TPU staffmemo.txt/READ_ONLY
```

The /NOREAD_ONLY qualifier is the same as the /WRITE qualifier. On exiting, EVE writes out the main (or first) buffer to a file if the buffer has been modified. If necessary, EVE prompts you for the output file name.

In EVE, the status line shows whether the buffer is read-only or write. Also, you can change the read/write and modification attributes of the buffer during your editing session by using the SET BUFFER command.

/RECOVER
/NORECOVER (default)

Determines whether VAXTPU recovers your edits after a system failure by reading a journal from the interrupted session. (See the description of the /JOURNAL qualifier.)

There are two ways to recover your edits, depending on the type of journaling you used:

- If you used **buffer-change journaling**, which is the EVE default, you recover one buffer at time and can recover buffers from different editing sessions. For example, the following command invokes EVE to recover the text of the file JABBER.TXT:

```
$  EDIT/TPU jabber.txt/RECOVER
```

This is the same as invoking EVE and using the following command:

```
Command:  RECOVER BUFFER jabber.txt
```

If there is more than one buffer-change journal with the same name—for example, you may have two or more MAIN.TPU$JOURNAL files from different editing sessions—the recovery uses the highest version number available. To recover several text buffers (one after another), use the RECOVER BUFFER ALL command.

Recovery with a buffer-change journal restores only your text—it does not restore settings, key definitions, and other customizations done during the editing session, and it does not restore the contents of the Insert Here buffer or other system buffers. The recovery is usually quite fast. New text or other changes are then journaled.

The recovery does not re-create deleted files. If you deleted or renamed the source file associated with a buffer-change journal, the recovery fails. The source file is either the file initially read into the buffer (if any), or the last version of the file written from the buffer before the system failure.

- If you used **keystroke journaling**, you recover your editing session by reissuing the command for the original, aborted editing session—including all qualifiers—and adding the /RECOVER qualifier. For example, the following command recovers your edits using a keystroke journal named MYSESSION.TJL:

```
$  EDIT/TPU/JOURNAL=mysession
            .
            .
            .
***  system failure  ***
            .
            .
            .
$  EDIT/TPU/JOURNAL=mysession/RECOVER
```

EVE then recovers your editing session in a "player piano" fashion. Typically, after the recovery you exit to save the recovered text.

Keystroke journaling does not work on DECwindows and has other restrictions, as follows. These restrictions do *not* apply to buffer-change journaling.

- To recover your edits with a keystroke journal, all relevant files must be in the same state as at the start of the editing session being recovered—including any files you wrote out (saved) before the system failure. Therefore, before doing the recovery, you should rename the saved versions or move them to a different directory, to ensure that the recovery uses the original versions of the files.

- Check that logical names for your section file, command file, and initialization file are defined as for the original editing session, and that the recovery will use the correct version of these files.

- Check that the following terminal settings are the same as when you began the original editing session, because they may affect how your keystrokes are replayed:

    Device_Type

Edit_mode
Eightbit
Page
Width

- Recovery with a keystroke journal may fail or may not work properly if you pressed Ctrl/C during the original editing session, because pressing Ctrl/C is not recorded in the keystroke journal. Therefore, when you recover your edits, an operation that was canceled with Ctrl/C is replayed without interruption; this is likely to affect how the remaining keystrokes are replayed.

- If you used EVE as a "kept" editor, the keystroke journal records ATTACH, DCL, and SPAWN commands in EVE, but does not record operations done in the other process or subprocess. If these other operations affected any files used in the original editing session—for example, if you spawned a subprocess from EVE and then purged, renamed, deleted, or modified any relevant files—the recovery may fail or may not work properly.

- If you used the EVE command DCL, the recovery may fail or may not work properly, particularly if you cut a file name from a directory list in the DCL buffer and pasted it into an EVE command line. The keystroke recovery replays the operations, but the directory list or the file name or file version may not be the same as in the original session.

For more information about journaling and recovery, see the *VMS EVE Reference Manual*, or use the online help in EVE and read the topic called Journal Files.

Note: **Although journaling and recovery are quite reliable, your last few edits before the system failure may be lost. The safest way to protect your work against a system failure is to write out your edits frequently—particularly during all-day editing sessions.**

/SECTION[=section-file] (default)
/NOSECTION

Determines the section file you want to use, if any. A section file contains, in binary form, key definitions, compiled procedures, global variables, and so on. Effectively, the section file is the VAXTPU application you run—whether a customized version of EVE or an application you have created.

VAXTPU assumes the section file is in SYS$SHARE. If your section file is stored elsewhere, specify the device (disk) and directory of the section file. You cannot use wildcards to specify the section file. You use one section file at a time. The default file type is TPU$SECTION.

The default section file is defined systemwide by the logical name TPU$SECTION, which specifies the standard EVE section file (EVE$SECTION.TPU$SECTION).

There are two ways to specify the section file you want to use:

- Define the logical name TPU$SECTION to specify the section file.

  This lets you use the same section file for all editing sessions, including when you invoke VAXTPU within MAIL. Your definition of the logical name overrides the systemwide default. You can put the definition in your LOGIN.COM file.

For example, the following commands define TPU$SECTION as a file named MYSEC.TPU$SECTION in your top-level, login directory, and then invoke VAXTPU using that section file instead of the standard EVE section file:

```
$  DEFINE TPU$SECTION sys$login:mysec
$  EDIT/TPU
```

* Use the /SECTION qualifier and specify the section file on the command line.

    This overrides any definition of the TPU$SECTION logical name, whether a definition of your own or the systemwide default. For example, the following command invokes VAXTPU, using a section file named MYSEC.TPU$SECTION in your top-level, login directory, instead of the standard EVE section file:

    ```
    $  EDIT/TPU/SECTION=sys$login:mysec
    ```

If you use the /NOSECTION qualifier, VAXTPU does not use any section file. This prevents even the default EVE interface from being used. VAXTPU will be virtually unusable unless you specify a command file with procedures and executable statements that set up a text-processing environment. Use /NOSECTION when you are creating your own application without using EVE as a base, or with the /NODISPLAY qualifier for batch editing. For example, the following command invokes VAXTPU without a section file, using a VAXTPU command file named USER_APPL.TPU:

```
$  EDIT/TPU/NOSECTION/COMMAND=user_appl/NODISPLAY
```

At startup, a section file, if one is being is used, is loaded before VAXTPU compiles and executes a command file (if any) and before EVE executes an initialization file (if any). Thus, procedures, settings, and key definitions in a command file or initialization file override those in a section file.

To create a section file, do either of the following:

* In EVE, use the SAVE EXTENDED EVE command. For example, the following command creates a section file named MYSEC.TPU$SECTION in your current, default directory:

    ```
    Command:  SAVE EXTENDED EVE mysec
    DISK$1:[USER]MYSEC.TPU$SECTION;1 created
    903 procedures, 1168 variables, 621 keys saved
    ```

* In a VAXTPU procedure or command file, use the SAVE built-in procedure, usually at the end of a command file. For example, the following statements create a section file named MYSEC.TPU$SECTION in your top-level, login directory and then exit from VAXTPU:

    ```
            .
            .
            .

    SAVE ("sys$login:mysec"); ! create the section file
    EXIT;                     ! done -- end of command file
    ```

A section file is cumulative: it saves the current key definitions and other customizations, and those already in the section file you are using (if any). In EVE, the section file saves the following:

Compiled procedures
Global settings ("attributes")
Key definitions and learn sequences
Menu definitions for the DECwindows interface

A section file usually does *not* save the following:

Margins, tab stops, and other buffer settings
Width or number of windows
Contents of system buffers, such as the Insert Here buffer

For more information about creating section files, see the *VAX Text Processing Utility Manual* or the *VMS EVE Reference Manual*, or use the online help in EVE and read the topic called Section Files.

/START_POSITION=(row[,column])

Determines the row and column where the cursor first appears in the main (or first) buffer. The way this qualifier is processed depends on the VAXTPU application you are running.

For EVE, the default start position is 1,1, which is the upper left corner of the main (or first) buffer—row 1, column 1. This does not affect the initial cursor position when you create other buffers during the editing session, and does not limit the buffer size.

Use the /START_POSITION qualifier to begin editing at a particular line (or row) or at a particular character position (or column), such as when you want to skip over a standard heading in a file. If a batch log file or error message tells you there is an error on a given line of a program, you can specify that line number as the starting row, so that when you edit the program source file, the cursor moves directly to that line. For example, the following command edits a file named TEST.COM, putting the cursor on line 10, column 5:

```
$ EDIT/TPU test.com/START_POSITION=(10,5)
```

If you simply want to start at a particular line in a file, you can omit the second parameter (the column) and you need not use parentheses. If you specify a line number greater than the number of lines in the file, EVE puts the cursor at the bottom of the buffer.

/WORK[=work-file]

Determines the work file that VAXTPU uses to swap memory for editing very large files. The work file is a temporary file, which is automatically deleted when you exit. Also, the work file is invisible—that is, it does not appear in the directory listing, although it does take up a file slot.

You cannot use wildcards to specify the work file. There is one work file per editing session. The default file type is TPU$WORK.

By default, VAXTPU creates a work file named TPU$WORK.TPU$WORK in SYS$SCRATCH, which is usually your top-level, login directory. There are two ways to specify a different work file:

• Define the logical name TPU$WORK.

This is useful if you want the work file created in an area other than SYS$SCRATCH, such as on a larger disk. You can put the definition in your LOGIN.COM file.

- Use the /WORK qualifier and specify the work file you want created.

  This overrides any definition of the TPU$WORK logical name. For example, the following command invokes VAXTPU, specifying the work file to be MYWORK.TPU$WORK:

  ```
  $  EDIT/TPU/WORK=mywork
  ```

To create the work file in an area other than SYS$SCRATCH, specify the device (disk) and directory of the work file.

/WRITE (default)
/NOWRITE

Determines whether exiting writes the main (or first) buffer to a file. This does not affect other buffers you create during the editing session.

The /WRITE qualifier is the same as the /NOREAD_ONLY qualifier. On exiting, EVE writes out the main (or first) buffer to a file if the buffer has been modified. If necessary, EVE prompts you for the output file name.

The /NOWRITE qualifier is the same as the /READ_ONLY qualifier. For EVE, this makes the main (or first) buffer write-locked and also makes it unmodifiable unless you use the /MODIFY qualifier. Use the /NOWRITE qualifier to view a file without making any changes. For example, the following command lets you view a file named STAFFMEMO.TXT so you can use cursor-movement commands but cannot change the text:

```
$  EDIT/TPU staffmemo.txt/NOWRITE
```

In EVE, the status line shows whether the buffer is read-only or write. Also, you can change the read/write and modification attributes of the buffer during your editing session by using the SET BUFFER command.

EXAMPLES

1 ```$ EDIT/TPU```

This example invokes VAXTPU. By default, this runs EVE, creating an empty buffer named Main. You can then begin typing and editing, or you can specify the file you want to create or edit by using the GET FILE, the OPEN, or the OPEN SELECTED command.

2 ```$ EDIT/TPU jabber.txt```

This command allows you to edit a file named JABBER.TXT in your current, default directory. If the file exists, EVE displays the text in the main window; if you are creating a new file, the main window is empty.

3   `$ EDIT/TPU *.txt`

> EVE lets you use logical names and wildcards to specify the input file. If more than one file matches your request, EVE shows a list of the matching files to choose from—in this case, a list of files with the type TXT in your current, default directory. If no file matches, EVE creates an empty buffer named Main.

4   `$ EDIT/TPU memo.txt/RECOVER`

> This example recovers the text of MEMO.TXT by using a buffer-change journal named MEMO_TXT.TPU$JOURNAL.

ENDSUBROUTINE

Defines the end of a subroutine in a command procedure. For more information about the ENDSUBROUTINE command, refer to the description of the CALL command.

FORMAT   **ENDSUBROUTINE**

EOD

EOD

Signals the end of a data stream when a command or program is reading data from an input device other than an interactive terminal.

FORMAT   **$ EOD**

PARAMETERS   *None.*

DESCRIPTION   The EOD (end of deck) command in a command procedure or in a batch job does the following:

- Terminates input data lines that begin with dollar signs ($). The DECK command indicates that the following lines begin with dollar signs and should be interpreted as data, not as commands; the EOD command indicates the end of the data lines.

- Terminates an input file if multiple input files are contained in the command stream without intervening commands. The program or command reading the data receives an end-of-file (EOF) condition when the EOD command is read.

The EOD command must be preceded by a dollar sign; the dollar sign must be in the first character position (column 1) of the input record.

EXAMPLES

1
```
$ CREATE WEATHER.COM
$ DECK
$ FORTRAN WEATHER
$ LINK WEATHER
$ RUN WEATHER
$ EOD
$ @WEATHER
```

In this example, the command procedure creates a command procedure called WEATHER.COM. The lines delimited by the DECK and EOD commands are written to the file WEATHER.COM. Then the command procedure executes WEATHER.COM.

2

```
$ EOJ
$ PRINT TESTDATA.OUT
...Second Input Data File...
$ EOD
...First Input Data File...
$ RUN MYPROG
$ PASSWORD HENRY
$ JOB HIGGINS
```
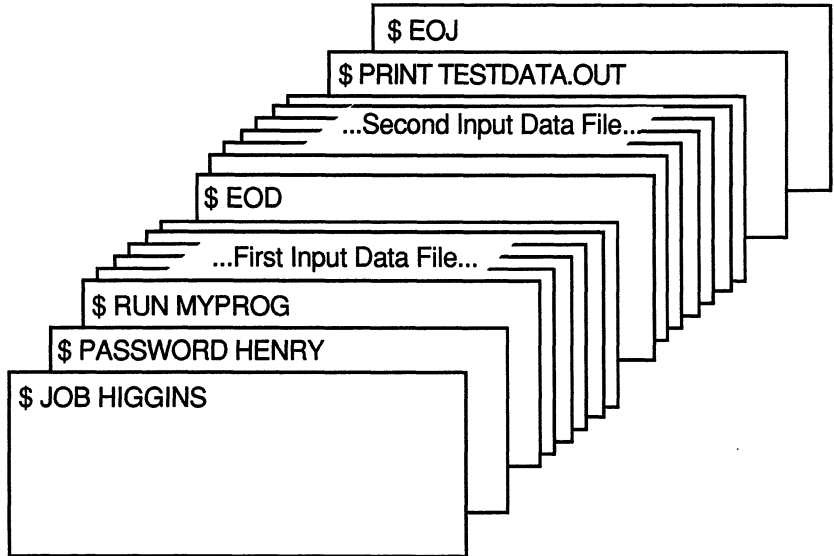
ZK-0785-GE

The program MYPROG requires two input files; these are read from the logical device SYS$INPUT. The EOD command signals the end of the first data file and the beginning of the second. The next line that begins with a dollar sign (a PRINT command in this example) signals the end of the second data file.

EOJ

Marks the end of a batch job submitted through a card reader.

FORMAT $ EOJ

PARAMETERS   *None.*

DESCRIPTION

The EOJ (end of job) command marks the end of a batch job submitted through a card reader. An EOJ card is not required; however, if present, the first nonblank character in the command line must be a dollar sign ($). If issued in any other context, the EOJ command logs the process out. The EOJ command cannot be abbreviated.

The EOF card is equivalent to the EOJ card.

EXAMPLE


```
$ EOJ
...Command Input Stream...
$ PASSWORD HENRY
$ JOB HIGGINS
```
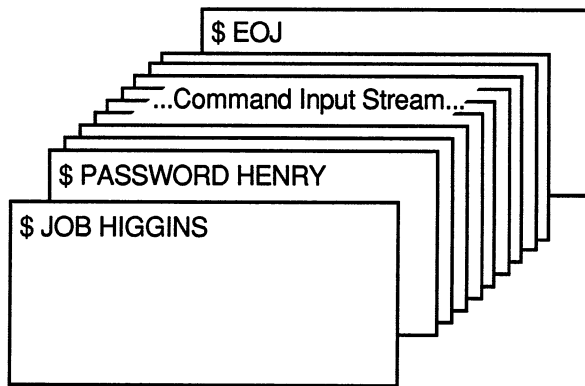
ZK-0786-GE

The JOB and PASSWORD commands mark the beginning of a batch job submitted through the card reader; the EOJ command marks the end of the job.

EXAMINE

Displays the contents of virtual memory.

Requires user-mode read (R) and write (W) access to the virtual memory location whose contents you want to examine.

FORMAT **EXAMINE** *location[:location]*

PARAMETER *location[:location]*
Specifies a virtual address or a range of virtual addresses (where the second address is larger than the first) whose contents you want to examine. If you specify a range of addresses, separate the beginning and ending addresses with a colon (:).

A location can be any valid arithmetic expression containing arithmetic or logical operators or previously assigned symbols. Radix qualifiers determine the radix in which the address is interpreted; hexadecimal is the initial default radix. Symbol names are always interpreted in the radix in which they were defined. The radix operators %X, %D, or %O can precede the location. A hexadecimal value must begin with a number (or be preceded by %X).

The DEPOSIT and EXAMINE commands maintain a pointer to the current memory location. The EXAMINE command sets this pointer to the last location examined when you specify an EXAMINE command. You can refer to this location using the period (.) in a subsequent EXAMINE command or DEPOSIT command.

DESCRIPTION The EXAMINE command displays the contents of virtual memory. The address is displayed in hexadecimal format and the contents are displayed in the radix requested, as follows:

```
address:  contents
```

If the address specified is not accessible to user mode, four asterisks (*) are displayed in the contents field.

Radix Qualifiers: The radix default for a DEPOSIT command or an EXAMINE command determines how the command interprets numeric literals. The initial default radix is hexadecimal; all numeric literals in the command line are assumed to be hexadecimal values. If a radix qualifier modifies an EXAMINE command, that radix becomes the default for subsequent EXAMINE and DEPOSIT commands, until another qualifier overrides it. For example:

```
$ EXAMINE/DECIMAL 900
00000384:  0554389621
```

The EXAMINE command interprets the location 900 as a decimal number and displays the contents of that location in decimal. All subsequent DEPOSIT and EXAMINE commands assume that numbers you enter for addresses and data are decimal. Note that the EXAMINE command always displays the address location in hexadecimal format.

Symbol names defined by = (assignment statement) commands are always interpreted in the radix in which they were defined.

Note that hexadecimal values entered as examine locations or as data to be deposited must begin with a numeric character (0 to 9). Otherwise, the command interpreter assumes that you have entered a symbol name, and attempts symbol substitution.

You can use the radix operators %X, %D, or %O to override the current default when you enter the EXAMINE command. For example:

```
$ EXAMINE/DECIMAL %X900
00000900:  321446536
```

This command requests a decimal display of the data in the location specified as hexadecimal 900.

Length Qualifiers: The initial default length unit for the EXAMINE command is a longword. The EXAMINE command displays data, 1 longword at a time, with blanks between longwords. If a length qualifier modifies the command, that length becomes the default length of a memory location for subsequent EXAMINE and DEPOSIT commands, until another qualifier overrides it.

Restriction on Placement of Qualifiers: The EXAMINE command analyzes expressions arithmetically. Therefore, qualifiers are interpreted correctly only when they appear immediately after the command name.

QUALIFIERS

/ASCII
Displays the data at the specified location in ASCII format.

Binary values that do not have ASCII equivalents are displayed as periods (.).

When you specify the /ASCII qualifier, or when ASCII mode is the default, hexadecimal is used as the default radix for numeric literals that are specified on the command line.

/BYTE
Displays data at the specified location 1 byte at a time.

/DECIMAL
Displays the contents of the specified location in decimal format.

/HEXADECIMAL
Displays the contents of the specified location in hexadecimal format.

/LONGWORD
Displays data at the specified location 1 longword at a time.

/OCTAL
Displays the contents of the specified location in octal format.

/WORD
Displays data at the specified location 1 word at a time.

EXAMPLES

1
```
$ RUN    MYPROG
[Ctrl/Y]
$ EXAMINE   2678
0002678:  1F4C5026
$ CONTINUE
```

> In this example, the RUN command begins execution of the image
> MYPROG.EXE. While MYPROG is running, pressing Ctrl/Y interrupts
> its execution, and the EXAMINE command displays the contents of virtual
> memory location 2678 (hexadecimal).

2
```
$ BASE = %X1C00
$ READBUF = BASE + %X50
$ ENDBUF = BASE + %XA0
$ RUN    TEST
[Ctrl/Y]
$ EXAMINE/ASCII READBUF:ENDBUF
00001C50:  BEGINNING OF FILE MAPPED TO GLOBAL SECTION
     .
     .
     .
```

> In this example, before executing the program TEST.EXE, symbolic names
> are defined for the program's base address and for labels READBUF
> and ENDBUF; all are expressed in hexadecimal format using the radix
> operator %X. READBUF and ENDBUF define offsets from the program
> base.
>
> While the program is executing, pressing Ctrl/Y interrupts it, and the
> EXAMINE command displays in ASCII format all data between the
> specified memory locations.

EXCHANGE

Invokes the Exchange Utility (EXCHANGE), which manipulates mass storage volumes that are written in formats other than those normally recognized by the VMS operating system.

EXCHANGE allows you to perform any of the following tasks:

- Create foreign volumes.
- Transfer files to and from the volume.
- List directories of the volume.

For block-addressable devices, such as RT–11 disks, EXCHANGE performs additional operations such as renaming and deleting files. EXCHANGE can also manipulate Files–11 files that are images of foreign volumes; these files are called **virtual devices**.

For a complete description of EXCHANGE, see the *VMS Exchange Utility Manual*.

FORMAT   **EXCHANGE**   *[subcommand] [filespec] [filespec]*

EXCHANGE/NETWORK

EXCHANGE/NETWORK

Enables the VMS operating system to transfer files to or from operating systems that do not support VMS file organizations. The transfer occurs over a DECnet network communications link that connects VMS and non-VMS operating system nodes.

Using DECnet services, the EXCHANGE/NETWORK command can perform any of the following tasks:

- Transfer files between a VMS node and a non-VMS system node.

- Transfer a group of input files to a group of output files.

- Transfer files between two non-VMS nodes, provided those nodes share DECnet connections with the VMS node that issues the EXCHANGE /NETWORK command.

FORMAT   **EXCHANGE/NETWORK**   *input-filespec[,...]*
output-filespec

PARAMETERS   *input-filespec[,...]*
Specifies the name of an existing file to be transferred. Wildcard characters are allowed. If you specify more than one file, separate the file specifications with commas (,).

output-filespec
Specifies the name of the output file into which the input is transferred.

You must specify at least one field in the output file specification. If you omit the device or directory, your current default device and directory are used. The EXCHANGE/NETWORK command replaces any other missing fields (file name, file type, and version number) with the corresponding field of the input file specification.

The EXCHANGE/NETWORK command creates a new output file for every input file that you specify.

You can use the asterisk (*) wildcard character in place of the file name, the file type, or the version number. The EXCHANGE/NETWORK command uses the corresponding field in the related input file to name the output file. You can also use the wildcard character in the output file specification to direct EXCHANGE/NETWORK to create more than one output file. For example:

```
$  EXCHANGE/NETWORK A.A,B.B  MYPC::*.C
```

This EXCHANGE/NETWORK command creates the files A.C and B.C at the non-VMS target node MYPC.

A more complete explanation of wildcard characters and version numbers follows in the Description section.

DCL1-213

DESCRIPTION   The EXCHANGE/NETWORK command imposes the following restrictions:

- Transfers of files can occur only between disk devices. (If a disk device is not the desired permanent residence for the file, you must either move the file to a disk before issuing the command or retrieve the file from a disk after the command completes.)

- The remote system must have a block size of 512 bytes, where a byte is 8 bits long.

- The nodes transferring files must support the DECnet Data Access Protocol (DAP).

The VMS Record Management Services (RMS) facility provides VMS access to records in VMS RMS files. To transfer VMS RMS files between two nodes where both nodes are VMS nodes, use one of the other DCL commands (such as COPY, APPEND, or CONVERT), as appropriate. These commands recognize RMS file organizations and are designed to ensure that RMS record structures are preserved as your files are moved.

Use the EXCHANGE/NETWORK command to transfer files between VMS nodes and non-VMS nodes when the differences in the file organizations would otherwise prevent the transfer or could lead to undesirable results. While using the COPY command ensures that both the contents and the attributes of a replicated file are preserved, the EXCHANGE/NETWORK command has more advantages. The EXCHANGE/NETWORK command offers you explicit control of your record attributes during file transfers, with the opportunity to make a file usable on several different operating systems.

The EXCHANGE/NETWORK command transfers files between VMS nodes and non-VMS nodes connected to the same DECnet network. If the non-VMS system does not support VMS file organizations, the EXCHANGE /NETWORK command can modify or discard file and record attributes during the transfer. However, if the target system is a VMS node, you have the option of applying new file and record attributes to the output file by supplying a File Definition Language (FDL) file, as described later in this section. The EXCHANGE/NETWORK command provides a number of defaults to handle the majority of transfers properly. However, in some situations, you need to know your file or record format requirements at both nodes.

VMS File and Record Attributes

All RMS files in the VMS environment include stored information, known as the file and record attributes, to describe the file and record characteristics. File attributes consist of items such as file organization, file protection, and file allocation information. Record attributes consist of items such as the record format, record size, key definitions for indexed files, and carriage control information. These attributes define the data format and access methods for the VMS RMS facility.

Non-VMS operating systems that do not support VMS file organizations have no means of storing file and record attributes with their files. Transferring a VMS file to a non-VMS system that is unable to store and handle file and record attributes can result in most of this information being discarded. Removing these attributes from a file can render it useless if it must be returned to the VMS system.

Transferring Files to VMS Nodes

When you transfer files to a VMS system from a non-VMS system, the files typically assume default file and record attributes. However, you can specify the attributes that you want the file to acquire in a File Definition Language (FDL) file. If you specify an FDL file with the /FDL qualifier, the FDL file determines the characteristics of the output file. This feature is useful in establishing compatible file and record attributes when you transfer a file from a non-VMS system to a VMS system. However, when you use an FDL file, you also assume responsibility for determining the required characteristics.

For more information on FDL files, see the *VMS File Definition Language Facility Manual*.

Transferring Files to Non-VMS Nodes

The EXCHANGE/NETWORK command discards file and record attributes associated with a VMS file during a transfer to a non-VMS system that does not support VMS file organizations. Be aware that the loss of file and record attributes in the transfer can render the output file useless for many applications.

Selecting Transfer Modes

The EXCHANGE/NETWORK command has four transfer mode options: AUTOMATIC, BLOCK, RECORD, and CONVERT. For most file transfers, AUTOMATIC is sufficient. The AUTOMATIC transfer mode option allows the EXCHANGE/NETWORK command to transfer files using either block or record I/O. The selection is based on the input file organization and the operating systems involved.

Selecting the BLOCK transfer mode option forces the EXCHANGE /NETWORK command to open both the input and output files for block I/O access. The input file is then transferred to the output file block by block. Use this transfer mode when you transfer executable images. It is also useful when you must preserve a file's content exactly, which is a common requirement when you store files temporarily on another system or when cooperating applications exist on the systems.

Selecting the RECORD transfer mode option forces the EXCHANGE /NETWORK command to open both the input file and output file for record I/O access. The input file is then transferred to the output file record by record. This transfer mode is primarily used for transferring text files.

Selecting the CONVERT transfer mode option forces the EXCHANGE /NETWORK command to open the input file for RECORD access and the output file for BLOCK access. Records are then read in from the input file, packed into blocks, and are written to the output file. This transfer mode is primarily used for transferring files with no implied carriage control.

For example, to transfer a file created with DIGITAL Standard Runoff (DSR) to a DECnet-DOS system, you must use the CONVERT transfer mode option. To transfer the resultant output file back to a VMS node, use the AUTOMATIC transfer mode option.

Wildcard Characters

Wildcard characters are permitted in the file specifications and follow the behavior typical of other VMS commands with respect to the VMS node.

When more than one input file is specified, but wildcards are not specified in the output file specification, the first input file is copied to the output file, and each subsequent input file is transferred and given a higher version number of the same output file name. Note that the files are not concatenated into a single output file. Also note that when you transfer files to foreign systems that do not support version numbers, only one output file results, and it is the last input file.

To create multiple output files, specify multiple input files and use at least one of the following:

- An asterisk wildcard character in the output file name, file type, or version number field

- Only a node name, a device name, or a directory specification as the output file specification

When you create multiple output files, the EXCHANGE/NETWORK command uses the corresponding field from each input file in the output file name.

Use the /LOG qualifier when you specify multiple input and output files to verify that the files were copied as you intended.

Version Numbers

The following guidelines apply when the target node file formats accept version numbers.

If no version numbers are specified for input and output files, the EXCHANGE/NETWORK command (by default) assigns a version number to the output files that is either of the following:

- The version number of the input file

- A version number one greater than the highest version number of an existing file with the same file name and file type

When the output file version number is specified by an asterisk wildcard character, the EXCHANGE/NETWORK command uses the version numbers of the associated input files as the version numbers of the output files.

If the output file specification has an explicit version number, the EXCHANGE/NETWORK command normally uses that number for the output file specification. However, if an equal or higher version of the output file already exists, no warning message is issued, the file is copied, and the version number is set to a value one greater than the highest version number already existing.

File Protection and Creation/Revision Dates

The EXCHANGE/NETWORK command treats an output file as a new file when any portion of the output file name is specified explicitly. When the output node is a VMS system, the creation date for a new file is set to the current time and date. However, if the output file specification consists *only* of wildcard characters, the output file no longer qualifies as a new file, and, therefore, the creation date of the input file is used. That is, if the output file specification is one of the following, the creation date becomes that of the input file: *, *.*, or *.*;*.

The revision date of the output file is always set to the current time and date; the backup date is set to zero. The output file is assigned a new expiration date. (Expiration dates are set by the file system if retention is enabled; otherwise, they are set to zero.)

When the target node is a VMS node, the protection and access control list (ACL) of the output file is determined by the following parameters, in the following order:

1   Protection of previously existing versions of the output file

2   Default protection and ACL of the output directory

3   Process default file protection

For an introduction to ACLs, see the *Guide to VMS System Security* and the *VMS DCL Concepts Manual*.

On VMS systems, the owner of the output file usually is the same as the creator of the output file. However, if a user with extended privileges creates the output file, the owner is either the owner of the parent directory or the owner of a previous version of the output file, if one exists.

Extended privileges include any of the following:

* SYSPRV (system privilege) or BYPASS

* System user identification code (UIC)

* GRPPRV (group privilege) if the owner of the parent directory (or previous version of the output file) is in the same group as the creator of the new output file

* An identifier (with the resource attribute) representing the owner of the parent directory (or previous version of the output file)

QUALIFIERS   */BACKUP*
Modifies the time value specified with the /BEFORE or the /SINCE qualifier. The /BACKUP qualifier selects files according to the dates of their most recent backups. This qualifier is incompatible with the /CREATED, /EXPIRED, and /MODIFIED qualifiers, which also allow you to select files according to time attributes. If you do not specify any of these four time qualifiers, the default is the /CREATED qualifier.

/BEFORE[=time]

Selects only those files dated prior to the specified time. You can specify time as an absolute time, as a combination of absolute and delta times, or as one of the following keywords: TODAY (default), TOMORROW, or YESTERDAY. Specify one of the following qualifiers with the /BEFORE qualifier to indicate the time attribute to be used as the basis for selection: /BACKUP, /CREATED (default), /EXPIRED, or /MODIFIED.

For complete information about specifying time values, see the *VMS DCL Concepts Manual*.

/BY_OWNER[=uic]

Selects only those files whose owner user identification code (UIC) matches the specified owner UIC. The default UIC is that of the current process.

Specify the UIC by using standard UIC format as described in the *VMS DCL Concepts Manual*.

/CONFIRM
/NOCONFIRM (default)

Controls whether a request is issued before each file transfer operation to confirm that the operation should be performed on that file. The following responses are valid:

YES	NO	QUIT
TRUE	FALSE	Ctrl/Z
1	0	ALL
	Return	

You can use any combination of uppercase and lowercase letters for word responses. Word responses can be abbreviated to one or more letters (for example, T, TR, or TRU for TRUE), but these abbreviations must be unique. Affirmative answers are YES, TRUE, and 1. Negative answers include: NO, FALSE, 0, and pressing the Return key. Entering QUIT or pressing Ctrl/Z indicates that you want to stop processing the command at that point. When you respond by entering ALL, the command continues to process, but no further prompts are given. If you type a response other than one of those in the list, DCL issues an error message and redisplays the prompt.

/CREATED (default)

Modifies the time value specified with the /BEFORE or the /SINCE qualifier. The /CREATED qualifier selects files based on their dates of creation. This qualifier is incompatible with the /BACKUP, /EXPIRED, and /MODIFIED qualifiers, which also allow you to select files according to time attributes. If you do not specify any of these four time qualifiers, the default is the /CREATED qualifier.

/EXCLUDE=(filespec[,...])

Excludes the specified files from the file transfer operation. You can include a directory but not a device in the file specification. Wildcard characters (* and %) are allowed in the file specification. However, you cannot use relative version numbers to exclude a specific version. If you specify only one file, you can omit the parentheses.

/EXPIRED

Modifies the time value specified with the /BEFORE or the /SINCE
qualifier. The /EXPIRED qualifier selects files according to their expiration
dates. (The expiration date is set with the SET FILE/EXPIRATION_DATE
command.) The /EXPIRED qualifier is incompatible with the /BACKUP,
/CREATED, and /MODIFIED qualifiers, which also allow you to select files
according to time attributes. If you do not specify any of these four time
qualifiers, the default is the /CREATED qualifier.

/FDL=fdl-filespec

Specifies that the output file characteristics are described in the File
Definition Language (FDL) file. Use this qualifier when you require
special output file characteristics. For more information about FDL files,
see the *VMS File Definition Language Facility Manual*.

Use of the /FDL qualifier implies that the transfer mode is block by block.
However, the transfer mode you specify with the /TRANSFER_MODE
qualifier prevails.

/LOG
/NOLOG (default)

Controls whether the EXCHANGE/NETWORK command displays the file
specifications of each file copied.

When you use the /LOG qualifier, the EXCHANGE/NETWORK command
displays the following for each copy operation:

- The file specifications of the input and output files

- The number of blocks or the number of records copied (depending on
  whether the file is copied on a block-by-block or record-by-record basis)

/MODIFIED

Modifies the time value specified with the /BEFORE or the /SINCE
qualifier. The /MODIFIED qualifier selects files according to the date
on which they were last modified. This time qualifier is incompatible with
the /BACKUP, /CREATED, and /EXPIRED qualifiers, which also allow
you to select files according to time attributes. If you do not specify any of
these four time qualifiers, the default is the /CREATED qualifier.

/SINCE[=time]

Selects only those files dated after the specified time. You can specify
time as an absolute time, as a combination of absolute and delta times,
or as one of the following keywords: TODAY (default), TOMORROW, or
YESTERDAY. Specify one of the following time qualifiers with the /SINCE
qualifier to indicate the time attribute to be used as the basis for selection:
/BACKUP, /CREATED (default), /EXPIRED, or /MODIFIED.

For complete information about specifying time values, see the *VMS DCL
Concepts Manual*.

/TRANSFER_MODE=option

Specifies the I/O method to be used in the transfer. This qualifier is useful
for all file formats. You can specify any one of the following options:

Option	Function
AUTOMATIC	Allows the EXCHANGE/NETWORK command to determine the appropriate transfer mode. This is the default transfer mode.
BLOCK	Opens both the input and output files for block I/O and transfers the files block by block.
CONVERT[=option[,...]]	Reads records from the input file, packs them into blocks, and writes them to the output file in block mode. The options listed in the following table determine what additional information is inserted during the transfer.
RECORD	Opens both the input and output files for record I/O and transfers the files record by record. The target system must support record operations, and the input file must be record oriented.

The following four options are available with the CONVERT transfer mode to control the insertion of special characters in the records:

Option	Function
CARRIAGE_CONTROL	Any carriage control information in the input file is interpreted, is expanded into actual characters, and is included with each record.
COUNTED	The length of each record, in bytes, is included at the beginning of the record. The length includes all FIXED_CONTROL, CARRIAGE_CONTROL, and RECORD_SEPARATOR information in each record.
FIXED_CONTROL	All variable length with fixed control record (VFC) information is written to the output file as part of the data. This information follows the record length information, if the COUNTED option was specified.
RECORD_SEPARATOR=separator	A 1- or 2-byte record separator is inserted between each record. Record separator characters are the last characters in the record. The three choices for separator characters are as follows: • CR: Specifies carriage return only. • LF: Specifies line feed only. • CRLF: Specifies carriage return and line feed.

EXAMPLES

1 $ EXCHANGE/NETWORK VMS_FILE.DAT FOO::FOREIGN_SYS.DAT

In this example, the EXCHANGE/NETWORK command transfers the file VMS_FILE.DAT located in the current default device and directory

to the file FOREIGN_SYS.DAT on the non-VMS node FOO. Because the /TRANSFER_MODE qualifier was not explicitly specified, the EXCHANGE /NETWORK command automatically determines whether the transfer method should be block or record I/O.

```
$ EXCHANGE/NETWORK/TRANSFER_MODE=BLOCK -
_$ FOO::FOREIGN_SYS.DAT VMS_FILE.DAT
```

In this example, the EXCHANGE/NETWORK command transfers the file FOREIGN_SYS.DAT from the non-VMS node FOO to the file VMS_ FILE.DAT in the current default device and directory. Block I/O is specified for the transfer mode.

```
$ EXCHANGE/NETWORK/FDL=VMS_FILE_DEFINITION.FDL -
_$ FOO::REMOTE_FILE.TXT  VMS_FILE.DAT
```

In this example, the EXCHANGE/NETWORK command transfers the file REMOTE_FILE.TXT on node FOO to the file VMS_FILE.DAT. The file attributes for the output file VMS_FILE.DAT are obtained from the File Definition Language (FDL) source file VMS_FILE_DEFINITION.FDL. For more information about creating FDL files, see the *VMS File Definition Language Facility Manual*. Because the qualifier /FDL is specified and the /TRANSFER_MODE qualifier is omitted, the transfer mode uses block I/O, by default.

```
$ EXCHANGE/NETWORK -
_$ /TRANSFER_MODE=CONVERT=(CARRIAGE_CONTROL,COUNTED, -
_$ RECORD_SEPARATOR=CRLF,FIXED_CONTROL) -
_$ PRINT_FILE.TXT  FOO::*
```

In this example, the EXCHANGE/NETWORK command transfers the file PRINT_FILE.TXT from the current default device and directory to the file PRINT_FILE.TXT on the non-VMS node FOO. The use of the CONVERT option with the /TRANSFER_MODE qualifier forces the input file to be read in record by record, modified as specified by the CONVERT options that follow, and written to the output file block by block. As many records as will fit are packed into the output blocks.

The CONVERT option CARRIAGE_CONTROL specifies that carriage control information is converted to ASCII characters and is inserted before the data or is appended to the record, depending on whether prefix control or postfix control, or both, are used. The CONVERT option FIXED_ CONTROL specifies that any fixed control information be translated to ASCII characters and be inserted at the beginning of the record. The CONVERT option RECORD_SEPARATOR=CRLF appends the two specified characters, carriage return and line feed, to the end of the record. The CONVERT option COUNTED specifies that the total length of the record must be counted (once the impact of all the previous convert options have been added), and the result is to be inserted at the beginning of the record, in the first 2 bytes.

EXIT

Terminates processing of a command procedure or subroutine and returns control to the next higher command level—either an invoking command procedure or interactive DCL. The EXIT command also terminates an image normally after a user enters Ctrl/Y (executing another image has the same effect).

FORMAT   **EXIT** *[status-code]*

PARAMETER   *status-code*

Defines a numeric value for the reserved global symbol $STATUS. You can specify the status-code parameter as an integer or an expression equivalent to an integer value. The value can be tested by the next outer command level. The low-order 3 bits of the value determine the value of the global symbol $SEVERITY.

If you specify a status code, DCL interprets the code as a condition code. Note that even numeric values produce warning, error, and fatal error messages, and that odd numeric values produce either no message or a success or informational message.

If you do not specify a status code, the current value of $STATUS is saved. When control returns to the outer command level, $STATUS contains the status of the most recently executed command or program.

DESCRIPTION   The EXIT and STOP commands both provide a way to terminate the execution of a procedure. The EXIT command terminates execution of the current command procedure and returns control to the calling command level. If you enter the EXIT command from a noninteractive process (such as a batch job), at command level 0, then the process terminates.

The STOP command returns control to command level 0, regardless of the current command level. If you execute the STOP command from a command procedure or from a noninteractive process (such as a batch job), the process terminates.

When a DCL command, user program, or command procedure completes execution, the command interpreter saves the condition code value in the global symbol $STATUS. If an EXIT command does not explicitly set a value for $STATUS, the command interpreter uses the current value of $STATUS to determine the error status.

The low-order 3 bits of the status value contained in $STATUS represent the severity of the condition. The reserved global symbol $SEVERITY contains this portion of the condition code. Severity values range from 0 to 4, as follows:

Value	Severity
0	Warning
1	Success
2	Error
3	Information
4	Severe (fatal) error

Note that the success and information codes have odd numeric values, and that warning and error codes have even numeric values.

When any command procedure exits and returns control to another level, the command interpreter tests the current value of $STATUS. If $STATUS contains an even numeric value and if its high-order bit is 0, the command interpreter displays the system message associated with that status code, if one exists. (If no message exists, the message NOMSG will be displayed.) If the high-order bit is 1, the message is not displayed.

When a command procedure exits following a warning or error condition that has already been displayed by a DCL command, the command interpreter sets the high-order bit of $STATUS to 1, leaving the remainder of the value intact. This ensures that error messages are not displayed by both the command that caused the error, and by the command procedure.

The EXIT command, when used after you interrupt an image with Ctrl/Y, causes a normal termination of the image that is currently executing. If the image declared any exit-handling routines, they are given control. This is in contrast to the STOP command, which does not execute exit-handling routines. For this reason, the EXIT command is generally preferable to the STOP command.

EXAMPLES

1   `$ EXIT 1`

The EXIT command in this example exits to the next higher command level, giving $STATUS and $SEVERITY a value of 1.

2
```
$ ON WARNING THEN EXIT
$ FORTRAN 'P1'
$ LINK 'P1'
$ RUN 'P1'
```

The EXIT command in this example is used as the target of an ON command; this statement ensures that the command procedure terminates whenever any warnings or errors are issued by any command in the procedure.

The procedure exits with the status value of the command or program that caused the termination.

3

```
$ START:
$          IF (P1 .EQS. "TAPE") .OR. (P1 .EQS. "DISK") THEN GOTO 'P1'
$          INQUIRE P1 "Enter device (TAPE or DISK)"
$          GOTO START
$ TAPE: !  Process tape files
   .
   .
   .
$          EXIT
$ DISK: ! Process disk files
   .
   .
   .
$          EXIT
```

The command procedure in this example shows how to use the EXIT command to terminate different command paths within the procedure. To execute the procedure, you must enter either TAPE or DISK as a parameter. The IF command uses a logical OR to test whether either of these strings was entered. If the result is true, the GOTO command branches to the corresponding label. If P1 was neither TAPE nor DISK, the INQUIRE command prompts for a correct parameter.

The commands following each of the labels TAPE and DISK provide different paths through the procedure. The EXIT command before the label DISK ensures that the commands after the label DISK are executed only if the procedure explicitly branches to DISK.

Note that the EXIT command at the end of the procedure is not required because the end of the procedure causes an implicit EXIT command. Use of the EXIT command, however, is recommended.

4

```
$ IF P1. EQS. "" THEN -
    INQUIRE P1 "Enter filespec (null to exit)"
$ IF P1 .EQS. "" THEN EXIT
$ PRINT 'P1'/AFTER=20:00/COPIES=50/FORMS=6
```

The command procedure in this example tests whether a parameter was passed to it; if the parameter was not passed, the procedure prompts for the required parameter. Then it retests the parameter P1. If a null string, indicated by a carriage return for a line with no data, is entered, the procedure exits. Otherwise, it executes the PRINT command with the current value of P1 as the input parameter.

5

```
$ IF P1 .EQS. "" THEN INQUIRE P1 "Code"
$ CODE = %X'P1'
$ EXIT CODE
```

The command procedure in this example, E.COM, illustrates how to determine the system message, if any, associated with a hexadecimal system status code. The procedure requires a parameter and prompts if none is entered. Then it prefixes the value with the radix operator %X and assigns this string to the symbol CODE. Finally, it issues the EXIT command with the hexadecimal value. The following example uses the procedure E.COM:

```
$  @E 1C
%SYSTEM-F-EXQUOTA, exceeded quota
```

When the procedure exits, the value of $STATUS is %X1C, which equates to the EXQUOTA message. Note that you can also use the F$MESSAGE lexical function to determine the message that corresponds to a status code.

6    $ RUN MYPROG
          `Ctrl/Y`
          $ EXIT

In this interactive example, the RUN command initiates execution of the image MYPROG.EXE. Then pressing Ctrl/Y interrupts the execution. The EXIT command that follows calls any exit handlers declared by the image before terminating MYPROG.EXE.

FONT

Converts an ASCII bitmap distribution format (BDF) into binary server natural form (SNF). The DECwindows server uses an SNF file to display a font. In addition to converting the BDF file to binary form, the font compiler provides statistical information about the font and the compilation process. For more information about using the font compiler, refer to the *VMS DECwindows Xlib Programming Volume*.

FORMAT   **FONT** *filespec*

GOSUB

Transfers control to a labeled subroutine in a command procedure without creating a new procedure level.

FORMAT   GOSUB *label*

PARAMETER   *label*

Specifies a label of 1 to 255 alphanumeric characters that appears as the first item on a command line. A label may not contain embedded blanks. When the GOSUB command is executed, control passes to the command following the specified label.

The label can precede or follow the GOSUB statement in the current command procedure. When you use a label in a command procedure, it must be terminated with a colon (:). If you use duplicate labels, control is always given to the label most recently read by DCL.

DESCRIPTION

Use the GOSUB command in command procedures to transfer control to a subroutine specified by the label. If the command stream is not being read from a random-access device (that is, a disk device), the GOSUB command performs no operation.

The RETURN command terminates the GOSUB subroutine procedure, returning control to the command following the calling GOSUB statement. The RETURN command accepts an optional status value.

The GOSUB command does not cause the creation of a new procedure level. Therefore, it is referred to as a "local" subroutine call. Any labels and local symbols defined in the current command procedure level are available to a subroutine invoked with a GOSUB command. The GOSUB command can be nested up to a maximum of 16 levels per procedure level.

When the command interpreter encounters a label, it enters the label in a label table. This table is allocated from space available in the local symbol table. If the command interpreter encounters a label that already exists in the table, the new definition replaces the existing one. Therefore, if you use duplicate labels, control is always given to the label most recently read by DCL. The following rules apply:

- If duplicate labels precede and follow the GOSUB command, control is given to the label preceding the command.

- If duplicate labels all precede the GOSUB command, control is given to the most recent label, that is, the one nearest the GOSUB command.

- If duplicate labels all follow the GOSUB command, control is given to the one nearest the GOSUB command.

If a label does not exist in the current command procedure, the procedure cannot continue and is forced to exit.

Note that the amount of space available for labels is limited. If a command procedure uses many symbols and contains many labels, the command interpreter may run out of table space and issue an error message.

EXAMPLE

```
$!
$! GOSUB.COM
$!
$ SHOW TIME
$ GOSUB TEST1
$ WRITE SYS$OUTPUT "success completion"
$ EXIT
$!
$! TEST1 GOSUB definition
$!
$ TEST1:
$     WRITE SYS$OUTPUT "This is GOSUB level 1."
$     GOSUB TEST2
$     RETURN %X1
$!
$! TEST2 GOSUB definition
$!
$ TEST2:
$     WRITE SYS$OUTPUT "This is GOSUB level 2."
$     GOSUB TEST3
$     RETURN
$!
$! TEST3 GOSUB definition
$!
$ TEST3:
$     WRITE SYS$OUTPUT "This is GOSUB level 3."
$     RETURN
```

This sample command procedure shows how to use the GOSUB command to transfer control to labeled subroutines. The GOSUB command transfers control to the subroutine labeled TEST1. The procedure executes the commands in subroutine TEST1, branching to the subroutine labeled TEST2. The procedure then executes the commands in subroutine TEST2, branching to the subroutine labeled TEST3. Each subroutine is terminated by the RETURN command. After TEST3 is executed, the RETURN command returns control back to the command line following each calling GOSUB statement. At this point, the procedure has been successfully executed.

GOTO

Transfers control to a labeled statement in a command procedure.

FORMAT	**GOTO** *label*

PARAMETER

label

Specifies a label of 1 to 255 alphanumeric characters that appears as the first item on a command line. A label cannot contain embedded blanks. When the GOTO command is executed, control passes to the command following the specified label.

When you use a label in a command procedure, it must be terminated with a colon (:). If you use duplicate labels, control is always given to the label most recently read by DCL.

DESCRIPTION

Use the GOTO command in command procedures to transfer control to a line that is not the next line in the procedure. The label can precede or follow the GOTO statement in the current command procedure. If the command stream is not being read from a random-access device (that is, a disk device), the GOTO command performs no operation.

If the target label of a GOTO command is inside a separate IF-THEN-ELSE construct, an error message (DCL-W-USGOTO) is returned.

When the command interpreter encounters a label, it enters the label in a label table. This table is allocated from space available in the local symbol table. If the command interpreter encounters a label that already exists in the table, the new definition replaces the existing one. Therefore, if you use duplicate labels, control is always given to the label most recently read by DCL. In general:

- If duplicate labels precede and follow the GOTO command, control is given to the label preceding the command.

- If duplicate labels all precede the GOTO command, control is given to the most recent label, that is, the one nearest the GOTO command.

- If duplicate labels all follow the GOTO command, control is given to the one nearest the GOTO command.

If a label does not exist in the current command procedure, the procedure cannot continue and is forced to exit.

Note that the amount of space available for labels is limited. If a command procedure uses many symbols and contains many labels, the command interpreter may run out of table space and issue an error message.

EXAMPLES

1
```
$ IF P1 .EQS. "HELP" THEN GOTO TELL
$ IF P1 .EQS. "" THEN GOTO TELL
    .
    .
    .
$ EXIT
$ TELL:
$ TYPE SYS$INPUT
To use this procedure, you must enter a value for P1.
    .
    .
    .
$ EXIT
```

> In this example, the IF command checks the first parameter passed to the command procedure; if this parameter is the string HELP or if the parameter is not specified, the GOTO command is executed and control is passed to the line labeled TELL. Otherwise, the procedure continues executing until the EXIT command is encountered. At the label TELL, a TYPE command displays data in the input stream that documents how to use the procedure.

2
```
$ ON ERROR THEN GOTO CHECK
    .
    .
    .
$ EXIT
$ CHECK:   ! Error handling routine
    .
    .
    .
$ END:
$ EXIT
```

> The ON command establishes an error-handling routine. If any command or procedure subsequently executed in the command procedure returns an error or severe error, the GOTO command transfers control to the label CHECK.

HELP

Displays information concerning use of the system, including formats and explanations of commands, parameters, and qualifiers.

FORMAT	**HELP** *[keyword[,...]]*

PARAMETER   *keyword[,...]*

Specifies one or more keywords that refer to the topic or subtopic on which you want information from a help library. To use the VMS HELP Facility in its simplest form, enter the HELP command from your terminal. The HELP Facility displays a list of topics at your terminal and the prompt Topic?. To see information on one of the topics, type the topic name after the prompt. The system displays information on that topic.

If the topic has subtopics, the HELP command lists the subtopics and displays the Subtopic? prompt. To get information on one of the subtopics, type the name after the prompt. To see information on another topic, press the Return key. You can now ask for information on another topic when the HELP Facility displays the Topic? prompt. Press the Return key to exit the HELP Facility and return to DCL command level.

DESCRIPTION   Information within help libraries is arranged in a hierarchical manner. The levels are as follows:

1   None—If you do not specify a keyword, the HELP Facility describes the HELP command and lists the topics that are documented in the root library. Each item in the list is a keyword in the first level of the hierarchy.

2   Topic-name—If you specify a keyword by naming a topic, the HELP Facility describes the topic as it is documented in either the root library or in one of the other enabled default libraries. Keywords for additional information available on this topic are listed.

3   Topic-name subtopic—If you specify a subtopic following a topic, the HELP Facility provides a description of the specified subtopic.

4   @filespec followed by any of the above—If you specify a help library to replace the current root library, the HELP Facility searches that library for a description of the topic or subtopic specified. The file specification must take the same form as the file specification included with the /LIBRARY command qualifier. However, if the specified library is an enabled user-defined default library, the file specification can be abbreviated to any unique leading substring of that default library's logical name translation.

If you use an asterisk (*) in place of any keyword, the HELP command displays all information available at the level that the asterisk replaces. For example, HELP COPY * displays all the subtopics under the topic COPY.

If you use an ellipsis (...) immediately after any primary keyword, the HELP Facility displays all the information on the specified topic and all subtopics of that topic. For example, HELP COPY ... displays information on the COPY topic as well as information on all the subtopics under COPY. The ellipsis can only be used from the topic level; it cannot be used from the subtopic level.

Wildcard characters (* and %) are allowed in the keyword.

QUALIFIERS

/INSTRUCTIONS (default)
/NOINSTRUCTIONS
Displays an explanation of the HELP command along with the list of topics (if no topic is specified). By default, the HELP command display includes a description of the facility and the format, along with the list of topics. If you specify the /NOINSTRUCTIONS qualifier, only the list of topics is displayed.

/LIBLIST (default)
/NOLIBLIST
Displays any auxiliary help libraries.

/LIBRARY=filespec
/NOLIBRARY
Uses an alternate help library instead of the default system library, SYS$HELP:HELPLIB.HLB. The specified library is used as the main (root) help library, and is searched for HELP Facility information before any user-defined default help libraries are checked.

If you omit the device and directory specification, the default is SYS$HELP, the logical name of the location of the system help libraries. The default file type is HLB.

The /NOLIBRARY qualifier excludes the default help library from the library search order.

/OUTPUT[=filespec]
/NOOUTPUT
Controls where the output of the command is sent. By default, the output is sent to SYS$OUTPUT, the current process default output stream or device.

If you enter the /OUTPUT qualifier with a partial file specification (for example, /OUTPUT=[JONES]), HELP is the default file name and LIS is the default file type. No wildcards are allowed.

If you enter the /NOOUTPUT qualifier, output is suppressed.

/PAGE (default)
/NOPAGE
Stops the display when the screen is full. You must press the Return key to continue.

If you specify the /NOPAGE qualifer, output continues until the information display ends or until you manually control the scrolling.

/PROMPT (default)
/NOPROMPT

Permits you to solicit further information interactively. If you specify the /NOPROMPT qualifier, the HELP Facility returns you to DCL command level after it displays the requested information.

If the /PROMPT qualifier is in effect, one of four different prompts is displayed, requesting you to specify a particular help topic or subtopic. Each prompt represents a different level in the hierarchy of help information. The four prompt levels are as follows:

1 Topic?—The root library is the main library and you are not currently examining the HELP Facility information for a particular topic.

2 [library-spec] Topic?—The root library is a library other than the main library and you are not currently examining the HELP Facility information for a particular topic.

3 [keyword] Subtopic?—The root library is the main library and you are currently examining the HELP Facility information for a particular topic (or subtopic).

4 A combination of 2 and 3.

When you encounter one of these prompts, you can enter any one of the responses described in the following table:

Response	Current Prompt Environment	Action
keyword[...]	1,2	Searches all enabled libraries for the keyword.
	3,4	Searches additional help libraries for the current topic (or subtopic) for the keyword.
@filespec keyword[...]	1,2	Same as above, except that the library specified by @filespec is now the root library. If the specified library does not exist, the HELP Facility treats @filespec as a normal keyword.
		Displays a list of topics available in the root library.
	3,4	Same as above; treats @filespec as a normal keyword.
		Displays the list of subtopics of the current topic (or subtopics) for which help exists.
Return	1	Exits from the HELP Facility.
	2	Changes root library to main library.
	3,4	Prompts for a topic or subtopic at the next higher level.
Ctrl/Z	1,2,3,4	Exits from the HELP Facility.

/USERLIBRARY=(level[,...])
/NOUSERLIBRARY

Names the levels of search for information in auxiliary libraries. The levels are as follows:

PROCESS	Libraries defined at process level
GROUP	Libraries defined at group level
SYSTEM	Libraries defined at system level
ALL	All libraries (default)
NONE	No libraries (same as the /NOUSERLIBRARY qualifier)

Auxiliary help libraries are libraries defined with the logical names HLP$LIBRARY, HLP$LIBRARY_1, HLP$LIBRARY_2, and so on. Libraries are searched for information in this order: root (current) library, main library (if not current), libraries defined at process level, libraries defined at group level, libraries defined at system level, and the root library. If the search fails, the root library is searched a second time so that the context is returned to the root library from which the search was initiated. The default is the /USERLIBRARY=ALL qualifier. If you specify only one level for the HELP Facility to search, you can omit the parentheses.

EXAMPLES

1
```
$ HELP
HELP
   .
   .   (HELP message text and list of topics)
   .
Topic?
```

In this example, the HELP command is entered without any qualifiers or parameters. This example produces a display of the help topics available from the root help library, SYS$HELP:HELPLIB.HLB.

If you enter one of the listed topics in response to the Topic? prompt, the HELP Facility displays information about that topic and a list of subtopics (if there are any). If one or more subtopics exist, the HELP Facility prompts you for a subtopic, as follows:

```
Topic?  ASSIGN
ASSIGN
   .
   .   (HELP message text and subtopics)
   .
ASSIGN Subtopic?
```

If you type a subtopic name, the HELP Facility displays information about that subtopic, as follows:

```
ASSIGN Subtopic?  Name
ASSIGN
  Name
   .
   .   (HELP message text and subtopics, if any)
   .
ASSIGN Subtopic?
```

If one or more sub-subtopics exist, the HELP Facility prompts you for a sub-subtopic; otherwise, as in the previous example, the facility prompts you for another subtopic of the topic you are currently inspecting.

Entering a question mark (?) redisplays the HELP Facility message and options at your current level. Pressing the Return key does either of the following:

- Moves you back to the previous help level if you are in a subtopic level.

- Terminates the HELP Facility if you are at the first level.

Pressing Ctrl/Z terminates the HELP Facility at any level.

2 `$ HELP COPY...`

The HELP command in this example displays a description of the COPY command and of the command's parameters and qualifiers. Note that the ellipsis can be used only from the topic level; it cannot be used from the subtopic level.

3
```
$ HELP/NOPROMPT ASSIGN/GROUP
    .
    .   (ASSIGN/GROUP HELP message)
    .
$
$ HELP/NOPROMPT/PAGE EDIT *
    .
    .   (HELP messages on all first-level EDIT subtopics)
    .
$
```

The two HELP commands request help on specific topics. In each case, the HELP command displays the help message you request and then returns you to DCL command level and the dollar sign prompt ($).

The first command requests help on the /GROUP qualifier of the ASSIGN command. The asterisk in the second example is a wildcard character. It signals the HELP Facility to display information about all EDIT subtopics, which are then displayed in alphabetical order. The /NOPROMPT qualifier suppresses prompting in both sample commands. The /PAGE qualifier on the second HELP command causes output to the screen to stop after each screen of information is displayed.

4
```
$ HELP FILL
Sorry, no documentation on FILL
Additional information available:
    .
    .   (list of first-level topics )
    .
Topic? @EDTHELP FILL
FILL
    .
    .   (FILL HELP message)
    .
@EDTHELP Topic?
```

When you enter a request for help on a topic that is not in the default help library, you can instruct the HELP Facility to search another help library for the topic. In this example, entering the command

HELP

@EDTHELP FILL instructs the HELP Facility to search the help library SYS$HELP:EDTHELP.HLB for information on FILL, an EDT editor command. The HELP Facility displays the message and prompts you for another EDT editor topic.

5
```
$ SET DEFAULT SYS$HELP
$ DEFINE HLP$LIBRARY EDTHELP
$ DEFINE HLP$LIBRARY_1 MAILHELP
$ DEFINE HLP$LIBRARY_2 BASIC
$ DEFINE HLP$LIBRARY_3 DISK2:[MALCOLM]FLIP
$ HELP REM
```

You can use logical names to define libraries for the HELP Facility to search automatically if it does not find the specified topic in the VMS root help library. This sequence of commands instructs the HELP Facility to search libraries in addition to the default root library, SYS$HELP:HELPLIB.HLB.

The four DEFINE statements create logical names for the four user-defined help libraries that the HELP Facility is to search after it has searched the root library. The first three entries are help libraries in the current default directory. By default, the HELP Facility searches for user-defined help libraries in the directory defined by the logical name SYS$HELP. The fourth entry is the help library FLIP.HLB in the directory DISK2:[MALCOLM]. Note that the logical names that you use to define these help libraries must be numbered consecutively; that is, you cannot skip any numbers.

The HELP Facility first searches the root library for REM. It then searches the libraries HLP$LIBRARY, HLP$LIBRARY_1, HLP$LIBRARY_2, and so on, until it finds REM or exhausts the libraries it knows it can search. When it finds REM in the BASIC.HLB library, the HELP Facility displays the appropriate help information and prompts you for a subtopic in that library. If you request information on a topic not in the BASIC.HLB library, the HELP Facility once again searches the help libraries you have defined.

IF

Tests the value of an expression and, depending on the syntax specified, executes the following:

- One command following the THEN keyword if the expression is true

- Multiple commands following the $THEN command if the expression is true

- One or more commands following the $ELSE command if the expression is false

---

FORMAT

$ IF   *expression THEN [$] command*

or

$ IF   *expression*

$ THEN   *[command]*

command

.
.
.

$ [ELSE]   *[command]*

command

.
.
.

$ ENDIF

PARAMETERS   ***expression***
Defines the test to be performed. The expression can consist of one or more numeric constants, string literals, symbolic names, or lexical functions separated by logical, arithmetic, or string operators.

Expressions in IF commands are automatically evaluated during the execution of the command. Character strings beginning with alphabetic characters that are not enclosed in quotation marks (" ") are assumed to be symbol names or lexical functions. The command language interpreter (CLI) replaces these strings with their current values.

Symbol substitution in expressions in IF commands is not iterative; that is, each symbol is replaced only once. However, if you want iterative substitution, precede a symbol name with an apostrophe (') or ampersand (&).

IF

The command interpreter does not execute an IF command when it contains an undefined symbol. Instead, the command interpreter issues a warning message and executes the next command in the procedure.

For a summary of operators and details on how to specify expressions, see the *VMS DCL Concepts Manual*.

command
Specifies the DCL command or commands to be executed, depending on the syntax specified, when the result of the expression is true or false.

DESCRIPTION

The IF command tests the value of an expression and executes a given command if the result of the expression is true. The expression is true if the result has an odd integer value, a character string value that begins with the letters Y, y, T, or t, or an odd numeric string value.

The expression is false if the result has an even integer value, a character string value that begins with any letter except Y, y, T, or t, or an even numeric string value.

EXAMPLES

1
```
$ COUNT = 0
$ LOOP:
$ COUNT = COUNT + 1
     .
     .
     .
$ IF COUNT .LE. 10 THEN GOTO LOOP
$ EXIT
```

This example shows how to establish a loop in a command procedure, using a symbol named COUNT and an IF statement. The IF statement checks the value of COUNT and performs an EXIT command when the value of COUNT is greater than 10.

2
```
$ IF P1 .EQS. "" THEN GOTO DEFAULT
$ IF (P1 .EQS. "A") .OR. (P1 .EQS. "B") THEN GOTO 'P1'
$ WRITE SYS$OUTPUT "Unrecognized parameter option ''P1' "
$ EXIT
$ A:       ! Process option a
  .
  .
  .
$ EXIT
$ B:       ! Process option b
  .
  .
  .
$ EXIT
$ DEFAULT: ! Default processing
  .
  .
  .
$ EXIT
```

This example shows a command procedure that tests whether a parameter was passed. The GOTO command passes control to the label specified as the parameter.

If the procedure is executed with a parameter, the procedure uses that parameter to determine the label to branch to. For example:

```
@TESTCOM A
```

When the procedure executes, it determines that P1 is not null, and branches to the label A. Note that the EXIT command causes an exit from the procedure before the label B.

8
```
$   SET NOON
    .
    .
    .
$   LINK CYGNUS,DRACO,SERVICE/LIBRARY
$ IF $STATUS
$ THEN
$   RUN CYGNUS
$ ELSE
$    WRITE SYS$OUTPUT "LINK FAILED"
$ ENDIF
$ EXIT
```

This command procedure uses the SET NOON command to disable error checking by the command procedure. After the LINK command, the IF command tests the value of the reserved global symbol $STATUS. If the value of $STATUS indicates that the LINK command succeeded, then the program CYGNUS is run. If the LINK command returns an error status value, the command procedure issues a message and exits.

INITIALIZE

Formats a disk or magnetic tape volume and writes a label on the volume. At the end of initialization, the disk is empty except for the system files containing the structure information. All former contents of the disk are lost.

Requires VOLPRO (volume protection) privilege for most INITIALIZE command operations.

FORMAT

INITIALIZE *device-name[:] volume-label*

PARAMETERS

device-name[:]
Specifies the name of the device on which the volume to be initialized is physically mounted.

The device does not have to be allocated currently; however, allocating the device before initializing it is the recommended practice.

volume-label
Specifies the identification to be encoded on the volume. For a disk volume, you can specify a maximum of 12 alphanumeric characters; for a magnetic tape volume, you can specify a maximum of 6 alphanumeric characters. Letters are automatically changed to uppercase. Nonalphanumeric characters are not allowed in the volume-label specification on disk.

To use ANSI "a" characters on the volume label on magnetic tape, you must enclose the volume name in quotation marks (" "). For an explanation of ANSI "a" characters, see the description of the /LABEL qualifier.

DESCRIPTION

The default format for disk volumes in the VMS operating system is called the Files–11 On-Disk Structure Level 2. The default for magnetic tape volumes is based on Level 3 of the ANSI standard for magnetic tape labels and file structure for informational interchange (ANSI X3.27-1978).

The INITIALIZE command can also initialize disk volumes in the Files–11 On-Disk Structure Level 1 format.

You do not need special privileges to override logical protection on the following devices:

- A blank disk or magnetic tape volume; that is, a volume that has never been written

- A disk volume that is owned by your current user identification code (UIC) or by the UIC [0,0]

- A magnetic tape volume that allows write (W) access to your current UIC that was not protected when it was initialized

In all other cases, you must have the VOLPRO privilege to initialize a volume.

When the INITIALIZE command initializes a magnetic tape volume, it always attempts to read the volume. A blank magnetic tape can sometimes cause unrecoverable errors, such as the following:

- An invalid volume number error message:

  ```
  %INIT-F-VOLINV, volume is invalid
  ```

- A runaway magnetic tape (this frequently occurs with new magnetic tapes that have never been written or that have been run through verifying machines). You can stop a runaway magnetic tape only by setting the magnetic tape drive off line and by then putting it back on line.

If this type of unrecoverable error occurs, you can initialize successfully a magnetic tape by repeating the INITIALIZE command from an account that has the VOLPRO (volume protection) privilege and by specifying the following qualifier in the command:

/OVERRIDE=(ACCESSIBILITY,EXPIRATION)

This qualifier ensures that the INITIALIZE command does not attempt to verify any labels on the magnetic tape.

If you have the VOLPRO privilege, the INITIALIZE command initializes a disk without reading the ownership information. If you do not have the VOLPRO privilege, the INITIALIZE command checks the ownership of the volume before initializing the disk. A blank disk or a disk with an incorrect format can sometimes cause a fatal drive error. If a blank disk or a disk with an incorrect format causes this type of error, you can initialize a disk successfully by repeating the INITIALIZE command with the /DENSITY qualifier from an account that has the VOLPRO privilege.

Many of the INITIALIZE command qualifiers allow you to specify parameters that can maximize input/output (I/O) efficiency.

QUALIFIERS

/ACCESSED=number-of-directories
Affects Files–11 On-Disk Structure Level 1 disks *only*.

Specifies that, for disk volumes, the number of directories allowed in system space must be a value from 0 to 255. The default value is 3.

/BADBLOCKS=(area[,...])
Specifies, for disk volumes, faulty areas on the volume. The INITIALIZE command marks the areas as allocated so that no data is written in them.

Possible formats for area are as follows:

lbn[:count]	Logical block number (LBN) of the first block and optionally a block count beginning with the first block, to be marked as allocated
sec.trk.cyl[:cnt]	Sector, track, and cylinder of the first block, and optionally a block count beginning with the first block, to be marked as allocated

All media supplied by Digital and supported on the VMS operating system, except diskettes and TU58 cartridges, are factory formatted and contain bad block data. The Bad Block Locator Utility (BAD) or the diagnostic formatter EVRAC can be used to refresh the bad block data or to construct it for the media exceptions above. The /BADBLOCKS qualifier is necessary only to enter bad blocks that are not identified in the volume's bad block data.

DIGITAL Storage Architecture (DSA) disks (for example, disks attached to UDA-50 and HSC50 controllers) have bad blocks handled by the controller, and appear logically perfect to the file system.

For information on how to run BAD, see the *VMS Bad Block Locator Utility Manual*.

/CLUSTER_SIZE=number-of-blocks

Defines, for disk volumes, the minimum allocation unit, in blocks. The maximum size you can specify for a volume is one-hundredth the size of the volume; the minimum size you can specify is calculated with the following formula:

$$\frac{disk\ size(number\ of\ blocks)}{255 * 4096}$$

For Files–11 On-Disk Structure Level 2 disks, the cluster size default depends on the disk capacity; disks that are 50,000 blocks or larger have a default cluster size of 3, while those smaller than 50,000 blocks have a default value of 1.

For Files–11 On-Disk Structure Level 1 disks, the cluster size must always be 1.

/DATA_CHECK[=(option[,...])]

Checks all read and write operations on the disk. By default, no data checks are made. Specify one or both of the following options:

READ	Checks all read operations.
WRITE	Checks all write operations; default if only the /DATA_CHECK qualifier is specified.

To override the checking you specify at initialization for disks, enter a MOUNT command to mount the volume.

/DENSITY=density-value

The /DENSITY qualifier is not applicable to the TK50 tape device.

For diskette volumes that are to be initialized on RX02 or RX33 diskette drives, specifies the density at which the diskette is to be formatted.

RX02 dual-density diskette drives allow diskettes to be initialized at single or double density. RX33 diskette drives allow diskettes to be initialized at double density only. To specify single-density formatting of a diskette, specify the density value SINGLE. To specify double-density formatting of a diskette, specify the density value DOUBLE.

If you do not specify a density value for a diskette being initialized on a drive, the system leaves the volume at the density to which the volume was last formatted.

For magnetic tape volumes, specifies the density in bits per inch (bpi) at which the magnetic tape is to be written.

For magnetic tape volumes, the density value specified can be 800 bpi, 1600 bpi, or 6250 bpi, as long as the density is supported by the magnetic tape drive. If you do not specify a density value for a blank magnetic tape, the system uses a default density of the highest value allowed by the tape drive. If the drive allows 6250-, 1600-, and 800-bpi operation, the default density is 6250 bpi. If you do not specify a density value for a magnetic tape that has been previously written, the system uses the density of the first record on the volume. If the record is unusually short, the density value will not default.

Note: **Diskettes formatted in double density cannot be read or written by the console block storage device (an RX01 drive) of a VAX/780 until they have been reformatted in single density.**

RX33 diskettes cannot be read from or written to by RX50 disk drives. RX50 diskettes can be read from and written to by RX33 disk drives; they cannot be formatted by RX33 disk drives.

/DIRECTORIES=number-of-entries
Specifies, for disk volumes, the number of entries to preallocate for user directories. The number of entries must be an integer between 16 and 16000. The default value is 16.

/ERASE
/NOERASE (default)
Physically destroys deleted data by writing over it. Controls the data security erase (DSE) operation on the volume before initializing it. The /ERASE qualifier applies to Files–11 On-Disk Structure Level 2 disk and ANSI magnetic tape volumes, and is valid for magnetic tape devices that support the hardware erase function, such as TU78 and MSCP magnetic tapes.

If you specify the /ERASE qualifier, a DSE operation is performed on the volume. For disk devices, the ERASE volume attribute is set. In effect, each file on the volume is erased when it is deleted.

Note that the amount of time taken by the DSE operation depends on the volume size; the INITIALIZE/ERASE command is always slower than the INITIALIZE/NOERASE command.

/EXTENSION=number-of-blocks

Specifies, for disk volumes, the number of blocks to use as a default extension size for all files on the volume. The extension default is used when a file increases to a size greater than its initial default allocation during an update. For Files–11 On-Disk Structure Level 2 disks, the value for the *number-of-blocks* parameter can range from 0 to 65,535. The default value is 5. For Files–11 On-Disk Structure Level 1 disks, the value can range from 0 to 255.

In VMS, the default volume extension is used only if no different extension has been set for the file and no default extension has been set for the process by using the SET RMS_DEFAULT command.

/FILE_PROTECTION=code
Affects Files–11 On-Disk Structure Level 1 disks *only*.

Defines, for disk volumes, the default protection to be applied to all files on the volume.

Specify the code according to the standard syntax rules described in the *VMS DCL Concepts Manual*. Any attributes not specified are taken from the current default protection.

Note that this attribute is not used when the volume is being used on a VMS system, but is provided to control the process's use of the volume on RSX–11M systems. VMS systems always use the default file protection. Use the SET PROTECTION/DEFAULT command to change the default file protection.

/GROUP

Defines a group volume. The /GROUP qualifier applies protection of read (R), write (W), execute (E), and delete (D) access to all ownership categories unless the /GROUP qualifier is specified with the /NOSHARE qualifier, in which case the volume protection is RWED for all but the world category. The owner user identification code (UIC) of the volume defaults to your group number and a member number of 0.

/HEADERS=number-of-headers

Specifies, for disk volumes, the number of file headers to be allocated for the index file. The minimum and default value is 16. The maximum is the value set with the /MAXIMUM_FILES qualifier.

This qualifier is useful when you want to create a number of files and want to streamline the process of allocating space for that number of file headers. If you do not specify this qualifier, the file system dynamically allocates space as it is needed for new headers on the volume.

/HIGHWATER (default)
/NOHIGHWATER
Affects Files–11 On-Disk Structure Level 2 disks *only*.

Sets the file highwater mark (FHM) volume attribute, which guarantees that users cannot read data that they have not written. You cannot specify the /NOHIGHWATER qualifier for magnetic tape.

The /NOHIGHWATER qualifier disables FHM for a disk volume.

/INDEX=position

Specifies the location of the index file for the volume's directory structure. Possible positions are as follows:

BEGINNING   Beginning of the volume

MIDDLE   Middle of the volume (default)

END   End of the volume

BLOCK:*n*   Beginning of the logical block specified by *n*

/LABEL=option

Defines characteristics for the magnetic tape volume label, as directed by the included option. The available options are as follows:

- OWNER_IDENTIFIER:"(14 ANSI characters)"

  Allows you to specify the Owner Identifier field in the volume label. The field specified can accept up to 14 ANSI characters.

- VOLUME_ACCESSIBILITY:"character"

  Specifies the character to be written in the volume accessibility field of the VMS ANSI volume label VOL1 on an ANSI magnetic tape. The character may be any valid ANSI "a" character. This set of characters includes numeric characters, uppercase letters, and any one of the following nonalphanumeric characters:

  ! " % ' ( ) * + , - . / : ; < = > ?

  By default, the VMS operating system provides a routine that checks this field in the following manner.

  - If the magnetic tape was created on a version of the VMS operating system that conforms to Version 3 of ANSI, then this option must be used to override any character other than an ASCII space.

  - If a VMS protection is specified and the magnetic tape conforms to an ANSI standard that is later than Version 3, then this option must be used to override any character other than an ASCII 1.

  If you specify any character other than the default, you must specify the /OVERRIDE=ACCESSIBILITY qualifier on the INITIALIZE and MOUNT commands in order to access the magnetic tape.

/MAXIMUM_FILES=n

Restricts the maximum number of files that the volume can contain. The /MAXIMUM_FILES qualifier overrides the default value, which is calculated as follows:

$$\frac{volume\ size\ in\ blocks}{(cluster\ factor + 1) * 2}$$

The maximum size you can specify for any volume is as follows:

$$\frac{volume\ size\ in\ blocks}{(cluster\ factor + 1)}$$

The minimum value is 0. Note that the maximum can be increased only by reinitializing the volume.

Note: **The MAXIMUM_FILES qualifier does not reserve or create space for new file headers on a volume. The file system dynamically allocates space as it is needed for new headers.**

/MEDIA_FORMAT=[NO]COMPACTION

Controls whether data records are automatically compacted and blocked together on a TA90E tape drive. Data compaction and record blocking increase the amount of data that can be stored on a single tape cartridge.

Note that once data compaction or non-compaction has been selected for a given cartridge, that same status applies to the entire cartridge.

/OVERRIDE=(option[,...])

Requests the INITIALIZE command to ignore data on a magnetic tape volume that protects it from being overwritten. You can specify one or more of the following options:

ACCESSIBILITY	(For magnetic tapes only.) If the installation allows, this option overrides any character in the Accessibility field of the volume. The necessity of this option is defined by the installation. That is, each installation has the option of specifying a routine that the magnetic tape file system will use to process this field. By default, VMS provides a routine that checks this field in the following manner. If the magnetic tape was created on a version of VMS that conforms to Version 3 of ANSI, this option must be used to override any character other than an ASCII space. If a VMS protection is specified and the magnetic tape conforms to an ANSI standard that is higher than Version 3, this option must be used to override any character other than an ASCII 1. To use the ACCESSIBILITY option, you must have the user privilege VOLPRO or be the owner of the volume.
EXPIRATION	(For magnetic tapes only.) Allows you to write to a tape that has not yet reached its expiration date. You may need to do this for magnetic tapes that were created before VMS Version 4.0 on Digital operating systems using the D% format in the volume Owner Identifier field. You must have the user privilege VOLPRO to override volume protection, or your UIC must match the UIC written on the volume.
OWNER_IDENTIFIER	Allows you to override the processing of the Owner Identifier field of the volume label.

If you specify only one option, you can omit the parentheses.

To initialize a volume that was initialized previously with the /PROTECTION qualifier, your UIC must match the UIC written on the volume or you must have VOLPRO privilege.

/OWNER_UIC=uic

Specifies an owner user identification code (UIC) for the volume. The default is your default UIC. Specify the UIC using standard UIC format as described in the *VMS DCL Concepts Manual*.

For magnetic tapes, no UIC is written unless protection on the magnetic tape is specified. If protection is specified, but no owner UIC is specified, your current UIC is assigned ownership of the volume.

/PROTECTION=(ownership[:access][,...])

Applies the specified protection to the volume. Specify the *ownership* parameter as system (S), owner (O), group (G), or world (W) and the *access* parameter as read (R), write (W), execute (E), or delete (D). The default is your default protection. Note that the /GROUP, /SHARE, and /SYSTEM qualifiers can also be used to define protection for disk volumes.

For magnetic tape, the protection code is written to a VMS-specific volume label. The system applies only read (R) and write (W) access restrictions; execute (E) and delete (D) access are meaningless. Moreover, the system and the owner are always given both read (R) and write (W) access to magnetic tapes, regardless of the protection code you specify.

For more information on specifying protection code, see the *VMS DCL Concepts Manual*. Any attributes not specified are taken from the current default protection.

When you specify a protection code for an entire disk volume, the access type E (execute) indicates create access.

/SHARE (default)
/NOSHARE

Permits all categories of access by all categories of ownership. The /NOSHARE qualifier denies access to group (unless the /GROUP qualifier is also specified) and world processes.

/STRUCTURE=level

Specifies whether the volume should be formatted in Files–11 On-Disk Structure Level 1 or 2 (the default). Structure Level 1 is incompatible with the /DATA_CHECK and /CLUSTER_SIZE qualifiers. The default protection for a Structure Level 1 disk is full access to system, owner, and group, and read (R) access to all other users.

/SYSTEM

Requires a system UIC or SYSPRV (system privilege) privilege.

Defines a system volume. The owner UIC defaults to [1,1]. Protection defaults to complete access by all ownership categories, except that only system processes can create top-level directories.

/USER_NAME=name

Specifies a user name to be associated with the volume. The name must be 1 to 12 alphanumeric characters. The default is your user name.

/VERIFIED
/NOVERIFIED

Indicates whether the disk has bad block data on it. Use the /NOVERIFIED qualifier to ignore bad block data on the disk. The default is the /VERIFIED qualifier for disks with 4096 blocks or more and the /NOVERIFIED qualifier for disks with less than 4096 blocks.

INITIALIZE

/WINDOWS=n

Specifies the number of mapping pointers (used to access data in the file) to be allocated for file windows. The value can be an integer in the range of 7 to 80. The default is 7.

EXAMPLES

1 `$  INITIALIZE/USER_NAME=CPA $FLOPPY1 ACCOUNTS`

> Initializes the volume on $FLOPPY1, labels the volume ACCOUNTS, and gives the volume a user name of CPA.

2
```
$  ALLOCATE DMA2:  TEMP
_DMA2: ALLOCATED
$  INITIALIZE  TEMP:  BACK_UP_FILE
$  MOUNT  TEMP:  BACK_UP_FILE
%MOUNT-I-MOUNTED, BACK_UP_FILE mounted on _DMA2:
$  CREATE/DIRECTORY  TEMP:[ARCHIE]
```

> The previous sequence of commands shows how to initialize an RK06/RK07 volume. First, the device is allocated, to ensure that no one else can access it. Then, when the volume is physically mounted on the device, the INITIALIZE command initializes it. When the volume is initialized, the MOUNT command makes the file structure available. Before you can place any files on the volume, you must create a directory, as shown by the CREATE/DIRECTORY command.

3
```
$  ALLOCATE MT:
_MTB1:  ALLOCATED
$  INITIALIZE MTB1:  SOURCE
$  MOUNT MTB1:  SOURCE
%MOUNT-I-MOUNTED, SOURCE mounted on _MTB1:
$  COPY *.FOR  MTB1:
$  DIRECTORY MTB1:
      .
      .
      .
$  DISMOUNT MTB1:
```

> These commands show the procedure necessary to initialize a magnetic tape. After allocating a drive, the magnetic tape is loaded on the device, and the INITIALIZE command writes the label SOURCE on it. Then, the MOUNT command mounts the magnetic tape so that files can be written on it.

4
```
$  BACKUP filespec MUA0: ... /MEDIA_FORMAT=NOCOMPACTION-
_$ /REWIND
```

> This example creates a BACKUP tape with compaction and record blocking disabled.

INITIALIZE/QUEUE

Creates or initializes queues. You use this command to create queues and to assign them names and attributes. The /QUEUE qualifier is required. The /BATCH qualifier is required to create a batch queue.

Requires OPER (operator) privilege.

FORMAT

INITIALIZE/QUEUE *queue-name[:]*

PARAMETER

queue-name[:]

Specifies the name of an execution queue or a generic queue. The queue name may be a string of 1 to 31 characters. The character string can include any uppercase and lowercase letters, digits, the dollar sign ($), and the underscore (_), and must include at least one alphabetic character.

DESCRIPTION

Initializing a Queue

You use the INITIALIZE/QUEUE command to create a queue or to change the attributes of an existing queue that is stopped. Normally you create output and batch queues by entering the necessary INITIALIZE/QUEUE commands in a site-specific system startup command procedure. However, once the system is running, you can use the INITIALIZE/QUEUE command to create additional queues as they are needed. You can use the INITIALIZE/QUEUE, START/QUEUE, and SET QUEUE commands to change queue attributes; however, you only can use the INITIALIZE and START commands on stopped queues.

To change attributes on a running queue, use the SET QUEUE command. To change attributes on a queue that cannot be altered with the SET QUEUE command, use the following procedure:

1   Stop the queue with the STOP/QUEUE/NEXT or the STOP/QUEUE /RESET command.

2   Restart the queue with the START/QUEUE or the INITIALIZE /QUEUE command.

To initialize an existing queue, use the following procedure:

1   Stop the queue with the STOP/QUEUE/NEXT command.

2   Initialize the queue.

3   Restart the queue.

Once a queue has been stopped, you can specify new qualifiers to replace existing queue attributes. Any qualifiers that you do not specify remain as they were when the queue was previously initialized, started, or set.

To initialize and start the queue at the same time, you can use the INITIALIZE/QUEUE/START command. If you want to initialize the queue only and start it at another time, you can enter only the INITIALIZE /QUEUE command. Later you can enter the START/QUEUE command to begin queue operations.

Note that initializing an existing queue does not delete any current jobs in that queue. Any new queue settings established by the new INITIALIZE /QUEUE command affect all jobs waiting in the queue or subsequently entering the queue. Any jobs that are executing in the queue when it is stopped complete their execution under the old settings.

The following qualifiers apply to generic and execution queues:

> /OWNER_UIC
> /PROTECTION
> /[NO]RETAIN
> /[NO]START

The following qualifiers apply to all types of execution queues:

> /BASE_PRIORITY
> /[NO]CHARACTERISTICS
> /[NO]ENABLE_GENERIC
> /ON
> /WSDEFAULT
> /WSEXTENT
> /WSQUOTA

Qualifiers that apply only to batch execution queues are as follows:

> /CPUDEFAULT
> /CPUMAXIMUM
> /[NO]DISABLE_SWAPPING
> /JOB_LIMIT

Qualifiers that apply only to printer, terminal, or server execution queues are as follows:

> /[NO]BLOCK_LIMIT
> /[NO]DEFAULT
> /FORM_MOUNTED
> /[NO]LIBRARY
> /[NO]PROCESSOR
> /[NO]RECORD_BLOCKING
> /[NO]SEPARATE

The /[NO]GENERIC qualifier distinguishes a generic queue from an execution queue.

You can use the /TERMINAL qualifier only with generic terminal queues.

Types of Queues

There are several different types of queues on the system. In general, queues can be divided into two major classes: generic and execution. When a job is sent to an execution queue, it is executed in that queue. No processing takes place in generic queues. Generic queues hold jobs that will execute on an execution queue when one is available.

The following are several types of generic queues:

Generic batch queue	Holds batch jobs for execution on batch execution queues.
Generic output queue	Holds jobs for execution on output queues. There are three types of generic output queues:

Generic printer queue	Holds print jobs for printing on output execution queues.
Generic server queue	Holds jobs for processing on output execution queues.
Generic terminal queue	Holds print jobs for printing on output execution queues.

The system manager or operator uses the /GENERIC qualifier to specify which execution queues can be accessed by a generic queue. You can use the /ENABLE_GENERIC qualifier when initializing an execution queue. This qualifier enables a generic queue to place jobs in an execution queue even if you did not specify that execution queue name with the /GENERIC qualifier of the generic queue.

The following are several types of execution queues:

Batch execution queue	Executes batch jobs.
Output execution queue	Processes print output jobs. There are three types of output execution queues:

Printer execution queue	Invokes a symbiont to process print jobs for a printer.
Server execution queue	Invokes a customer-written symbiont to process jobs.
Terminal execution queue	invokes a symbiont to process print jobs for a terminal printer.

Batch execution queues execute batch jobs. Batch jobs request the execution of one or more command procedures in a batch process.

Output execution queues process print jobs. A print job requests the processing of one or more files by a symbiont executing in a symbiont process. The default system symbiont is designed to print files on hardcopy devices (printers or terminals). Customer-written symbionts can be designed for this or any other file processing activity. Server queues process jobs using the server processor specified with the /PROCESSOR qualifier. Server queue processors are written by the customer.

Another type of queue is the logical queue. A logical queue is a special type of generic queue that can place work only into the execution queue specified in the ASSIGN/QUEUE command. The logical queue's relation to an execution queue remains in effect until you enter a DEASSIGN /QUEUE command to negate or change the assignment.

QUALIFIERS

/BASE_PRIORITY=n

Specifies the base process priority at which jobs are initiated from a batch execution queue. By default, if you omit the qualifier, jobs are initiated at the same priority as the base priority established by DEFPRI at system generation (usually 4). The base priority specifier can be any decimal value from 0 to 15.

You also can specify this qualifier for an output execution queue. In this context the /BASE_PRIORITY qualifier establishes the base priority of the symbiont process when the symbiont process is created.

/BATCH
/NOBATCH (default)

Specifies that you are initializing a batch queue. If you are reinitializing an existing queue, you can use the /BATCH qualifier only if the queue was created as a batch queue.

A batch queue is classified as either an execution queue or a generic queue. By default, the /BATCH qualifier initializes an execution queue. To specify a generic batch queue, use the /GENERIC qualifier together with the /BATCH qualifier.

The /BATCH and /DEVICE qualifiers are mutually exclusive; the /NOBATCH and /NODEVICE qualifiers cannot be used together.

/BLOCK_LIMIT=([lowlim,]uplim)
/NOBLOCK_LIMIT (default)

Limits the size of print jobs that can be processed on an output execution queue. The /BLOCK_LIMIT qualifier allows you to reserve certain printers for certain size jobs. You must specify at least one of the parameters.

The *lowlim* parameter is a decimal number referring to the minimum number of blocks accepted by the queue for a print job. If a print job is submitted that contains fewer blocks than the *lowlim* value, the job remains pending until the block limit for the queue is changed. After the block limit for the queue is decreased sufficiently, the job is processed.

The *uplim* parameter is a decimal number referring to the maximum number of blocks that the queue accepts for a print job. If a print job is submitted that exceeds this value, the job remains pending until the block limit for the queue is changed. After the block limit for the queue is increased sufficiently, the job is processed.

If you specify only an upper limit for jobs, you can omit the parentheses. For example, /BLOCK_LIMIT=1000 means that only jobs with 1000 blocks or less are processed in the queue. To specify only a lower job limit, you must use two consecutive quotation marks ("") to indicate the upper specifier. For example, /BLOCK_LIMIT=(500,"") means any job with 500 or more blocks is processed in the queue. You can specify both a lower and upper limit. For example, /BLOCK_LIMIT=(200,2000) means that jobs

with less than 200 blocks or more than 2000 blocks are not processed in the queue.

The /NOBLOCK_LIMIT qualifier cancels the previous setting established by the /BLOCK_LIMIT qualifier for that queue.

/CHARACTERISTICS=(characteristic[,...])
/NOCHARACTERISTICS (default)

Specifies one or more characteristics for processing jobs on an execution queue. If you specify only one characteristic, you can omit the parentheses. If a queue does not have all the characteristics that have been specified for a job, the job remains pending. Each time you specify the /CHARACTERISTICS qualifier, all previously set characteristics are cancelled. Only the characteristics specified with the qualifier are established for the queue.

Queue characteristics are installation specific. The *characteristic* parameter can be either a value from 0 to 127 or a characteristic name that has been defined by the DEFINE/CHARACTERISTIC command.

The /NOCHARACTERISTICS qualifier cancels any settings previously established by the /CHARACTERISTICS qualifier for that queue.

/CLOSE

Prevents jobs from being entered in the queue through PRINT or SUBMIT commands or as a result of requeue operations. To allow jobs to be entered, use the /OPEN qualifier. Whether a queue accepts or rejects new job entries is independent of the queue's state (such as paused, stopped, or stalled). When a queue is marked closed, jobs executing continue to execute. Jobs pending in the queue continue to be candidates for execution.

/CPUDEFAULT=time

Defines the default CPU time limit for all jobs in this batch execution queue. You can specify time as delta time, 0, INFINITE, or NONE (default). You can specify up to 497 days of delta time.

If the queue does not have a specified CPUMAXIMUM time limit and the value established in the user authorization file (UAF) has a specified CPU time limit of NONE, either the value 0 or the keyword INFINITE allows unlimited CPU time. If you specify NONE, the CPU time value defaults to the value specified either in the UAF or by the SUBMIT command (if included). CPU time values must be greater than or equal to the number specified by the SYSGEN parameter PQL_MCPULM. The time cannot exceed the CPU time limit set by the /CPUMAXIMUM qualifier. For information on specifying delta time, see the *VMS DCL Concepts Manual* or the *VMS User's Manual*. For more information on specifying CPU time limits, see Table DCL1-1.

/CPUMAXIMUM=time

Defines the maximum CPU time limit for all jobs in a batch execution queue. You can specify time as delta time, 0, INFINITE, or NONE (default). You can specify up to 497 days of delta time.

The /CPUMAXIMUM qualifier overrides the time limit specified in the user authorization file (UAF) for any user submitting a job to the queue. Either the value 0 or the keyword INFINITE allows unlimited CPU time. If you specify NONE, the CPU time value defaults to the value specified

either in the UAF or by the SUBMIT command (if included). CPU time values must be greater than or equal to the number specified by the SYSGEN parameter PQL_MCPULM.

For information on specifying delta times, see the *VMS DCL Concepts Manual* or the *VMS User's Manual*. For more information on specifying CPU time limits, see Table DCL1–1.

A CPU time limit for processes is specified by each user record in the system UAF. You also can specify the following: a default CPU time limit or a maximum CPU time limit for all jobs in a given queue, or a default CPU time limit for individual jobs in the queue. Table DCL1–1 shows the action taken for each value specified and possible combinations of specifications.

Table DCL1–1   CPU Time Limit Specifications and Actions

CPU Time Limit Specified by the SUBMIT Command?	Default CPU Time Limit Specified for the Queue?	Maximum CPU Time Limit Specified for the Queue?	Action Taken
No	No	No	Use the UAF value.
Yes	No	No	Use the smaller of SUBMIT command and UAF values.
Yes	Yes	No	Use the smaller of SUBMIT command and UAF values.
Yes	No	Yes	Use the smaller of SUBMIT command and queue's maximum values.
Yes	Yes	Yes	Use the smaller of SUBMIT command and queue's maximum values.
No	Yes	Yes	Use the smaller of queue's default and maximum values.
No	No	Yes	Use the maximum value.
No	Yes	No	Use the smaller of UAF and queue's default values.

/DEFAULT=(option[,...])
/NODEFAULT

Establishes defaults for certain options of the PRINT command. Defaults are specified by the list of options. If you specify only one option, you can omit the parentheses. After you set an option for the queue with the /DEFAULT qualifier, you do not have to specify that option in your PRINT command. If you do specify these options in your PRINT command, the values specified with the PRINT command override the values established for the queue with the /DEFAULT qualifier.

You cannot use the /DEFAULT qualifier with the /GENERIC qualifier.

Possible options are as follows:

[NO]BURST[=keyword]	Controls whether two file flag pages with a burst bar between them are printed preceding output. If you specify the value ALL (default), these flag pages are printed before each file in the job. If you specify the value ONE, these flag pages are printed once before the first file in the job.
[NO]FEED	Controls whether a form feed is inserted automatically at the end of a page.
[NO]FLAG[=keyword]	Controls whether a file flag page is printed preceding output. If you specify the value ALL (default), a file flag page is printed before each file in the job. If you specify the value ONE, a file flag page is printed once before the first file in the job.
FORM=type	Specifies the default form for an output execution queue. If a job is submitted without an explicit form definition, this form is used to process the job. See also the description of the /FORM_MOUNTED=type qualifier.
[NO]TRAILER[=keyword]	Controls whether a file trailer page is printed following output. If you specify the value ALL (default), a file trailer page is printed after each file in the job. If you specify the value ONE, a trailer page is printed once after the last file in the job.

When you specify the BURST option for a file, the [NO]FLAG option does not add or subtract a flag page from the two flag pages that are printed preceding the file.

For information on establishing mandatory queue attributes, see the description of the /SEPARATE qualifier. For information on specifying default queue attributes, see the *Guide to Maintaining a VMS System*.

/DESCRIPTION=string
/NODESCRIPTION (default)

Specifies a string of up to 255 characters used to provide operator-supplied information about the queue.

Enclose strings containing lowercase letters, blanks, or other nonalphanumeric characters (including spaces) in quotation marks (" ").

The /NODESCRIPTION qualifier removes any descriptive text that may be associated with the queue.

/DEVICE[=option]
/NODEVICE

Specifies that you are initializing an output queue of a particular type. If you are reinitializing an existing queue, you can use the /DEVICE qualifier only if the queue was created as an output queue. Possible options are as follows:

PRINTER   Indicates a printer queue.

SERVER   Indicates a server queue. A server queue is controlled by the user-modified or user-written symbiont specified with the /PROCESSOR qualifier.

TERMINAL   Indicates a terminal queue.

If you specify the /DEVICE qualifier without a queue type, the /DEVICE=PRINTER qualifier is used by default.

An output queue is classified as either an execution or generic queue. By default, the /DEVICE qualifier initializes an execution queue of the designated type. To specify a generic printer, server, or terminal queue, use the /GENERIC qualifier with the /DEVICE qualifier.

You specify the queue type with the /DEVICE qualifier for informational purposes. When an output execution queue is started, the symbiont associated with the queue determines the actual queue type. The standard symbiont examines device characteristics to establish whether the queue should be marked as printer or terminal. By convention, user-modified and user-written symbionts mark the queue as a server queue. The device type of a generic queue need not match the device type of its execution queues.

The /DEVICE and /BATCH qualifiers are mutually exclusive; the /NODEVICE and /NOBATCH qualifiers cannot be used together.

/DISABLE_SWAPPING
/NODISABLE_SWAPPING (default)

Controls whether batch jobs executed from a queue can be swapped in and out of memory.

/ENABLE_GENERIC (default)
/NOENABLE_GENERIC

Specifies whether files queued to a generic queue that does not specify explicit queue names with the /GENERIC qualifier can be placed in this execution queue for processing. For more information, see the description of the /GENERIC qualifier.

/FORM_MOUNTED=type

Specifies the mounted form for an output execution queue. If the stock of the mounted form does not match the stock of the default form, as indicated by the /DEFAULT=FORM qualifier, all jobs submitted to this queue without an explicit form definition enter a pending state. If a job is submitted with an explicit form and the stock of the explicit form is not identical to the stock of the mounted form, the job enters a pending state. In both cases, jobs remain pending until the stock of the mounted form of the queue is identical to the stock of the form associated with the job.

To specify the form type, use either a numeric value or a form name that has been defined by the DEFINE/FORM command. Form types are installation-specific. You cannot use the /FORM_MOUNTED qualifier with the /GENERIC qualifier.

/GENERIC[=(queue-name[,...])]
/NOGENERIC (default)

Specifies a generic queue. Also specifies that jobs placed in this queue can be moved for processing to compatible execution queues. The /GENERIC qualifier optionally accepts a list of target execution queues that have been previously defined. For a generic batch queue, these target queues must be batch execution queues. For a generic output queue, these target queues must be output execution queues, but can be of any type (printer, server, or terminal). For example, a generic printer queue can feed a mixture of printer and terminal execution queues.

If you do not specify any target execution queues with the /GENERIC qualifier, jobs can be moved to any execution queue that (1) is initialized with the /ENABLE_GENERIC qualifier, and (2) is the same type (batch or output) as the generic queue.

To define the queue as a generic batch or output queue, you use the /GENERIC qualifier with either the /BATCH or the /DEVICE qualifier. If you specify neither /BATCH nor /DEVICE on creation of a generic queue, the queue becomes a generic printer queue by default.

/JOB_LIMIT=n

Indicates the number of batch jobs that can be executed concurrently from the queue. Specify a number in the range 0 to 255. The job limit default value for n is 1.

/LIBRARY=file-name
/NOLIBRARY

Specifies the file name for the device control library. When you initialize an output execution queue, you can use the /LIBRARY qualifier to specify an alternate device control library. The default library is SYS$LIBRARY:SYSDEVCTL.TLB. You can use only a file name as the parameter of the /LIBRARY qualifier. The system always assumes that the file is located in SYS$LIBRARY and that the file type is TLB.

/ON=[node::]device[:] (printer, terminal, server queue)
/ON=node:: (batch queue)

Specifies the node or device, or both, on which this execution queue is located. For batch execution queues, you can specify only the node name. For output execution queues, you can include both the node name and the device name. By default, a queue executes on the same node from which you start the queue. The default device parameter is the same as the queue name.

The node name is used only in VAXcluster systems; it must match the node name specified by the SYSGEN parameter SCSNODE for the VAX computer on which the queue executes.

/OPEN (default)

Allows jobs to be entered in the queue through PRINT or SUBMIT commands or as the result of requeue operations. To prevent jobs from being entered in the queue, use the /CLOSE qualifier. Whether a queue accepts or rejects new job entries is independent of the queue's state (such as paused, stopped, or stalled).

/OWNER_UIC=uic

Enables you to change the user identification code (UIC) of the queue. Specify the UIC by using standard UIC format as described in the *VMS DCL Concepts Manual*. The default UIC is [1,4].

/PROCESSOR=file-name
/NOPROCESSOR

Allows you to specify your own print symbiont for an output execution queue. You can use any valid file name as a parameter of the /PROCESSOR qualifier. The system supplies the device and directory name SYS$SYSTEM and the file type EXE. If you use this qualifier for an output queue, it specifies that the symbiont image to be executed is SYS$SYSTEM:filename.EXE.

By default, SYS$SYSTEM:PRTSMB.EXE is the symbiont image associated with an output execution queue.

The /NOPROCESSOR qualifier cancels any previous setting established with the /PROCESSOR qualifier and causes SYS$SYSTEM:PRTSMB.EXE to be used.

/PROTECTION=(ownership[:access],...)

Specifies the protection of the queue. Specify the *ownership* parameter as system (S), owner (O), group (G), or world (W) and the *access* parameter as read (R), write (W), execute (E), or delete (D). A null access specification means no access. The default protection is (SYSTEM:E, OWNER:D, GROUP:R, WORLD:W). If you include only one protection code, you can omit the parentheses. For more information on specifying protection codes, see the *VMS DCL Concepts Manual*. For more information on controlling queue operations through UIC-based protection, see the *Guide to Maintaining a VMS System*.

/RECORD_BLOCKING (default)
/NORECORD_BLOCKING

Determines whether the symbiont can concatenate (or block together) output records for transmission to the output device. If you specify the /NORECORD_BLOCKING qualifier, the symbiont sends each formatted record in a separate I/O request to the output device. For the standard VMS print symbiont, record blocking can have a significant performance advantage over single-record mode.

/RETAIN[=option]
/NORETAIN (default)

Holds jobs in the queue in a retained status after they have executed. The /NORETAIN qualifier enables you to reset the queue to the default. Possible options are as follows:

| ALL (default) | Holds all jobs in the queue after execution. |
| ERROR | Holds in the queue only jobs that complete unsuccessfully. |

/SCHEDULE=SIZE (default)
/SCHEDULE=NOSIZE

Specifies whether pending jobs in an output execution queue are scheduled for printing based on the size of the job. When the default qualifier, /SCHEDULE=SIZE, is in effect, shorter jobs print before longer ones. When the /SCHEDULE=NOSIZE qualifier is in effect, jobs are printed in the order they were submitted, regardless of size.

If you enter this command while there are pending jobs in any queue, its effect on future jobs is unpredictable.

/SEPARATE=(option[,...])
/NOSEPARATE (default)

Specifies the mandatory queue attributes, or job separation options, for an output execution queue. Job separation options cannot be overridden by the PRINT command.

You cannot use the /SEPARATE qualifier with the /GENERIC qualifier.

The job separation options are as follows:

[NO]BURST	Specifies whether two job flag pages with a burst bar between them are printed at the beginning of each job.
[NO]FLAG	Specifies whether a job flag page is printed at the beginning of each job.
[NO]TRAILER	Specifies whether a job trailer page is printed at the end of each job.
[NO]RESET=(module[,...])	Specifies one or more device control library modules that contain the job reset sequence for the queue. The specified modules from the queue's device control library (by default SYS$LIBRARY:SYSDEVCTL) are used to reset the device each time a job reset occurs. The RESET sequence occurs after any file trailer and before any job trailer. Thus, all job separation pages are printed when the device is in its RESET state.

When you specify the /SEPARATE=BURST qualifier, the [NO]FLAG separation option does not add or subtract a flag page from the two flag pages that are printed preceding the job.

For information on establishing queue attributes that can be overridden, see the description of the /DEFAULT qualifier.

For more information on specifying mandatory queue attributes, see the *Guide to Maintaining a VMS System*.

/START
/NOSTART (default)

Starts the queue being initialized by the current INITIALIZE/QUEUE command.

/WSDEFAULT=n

Defines for a batch job a working set default, the default number of physical pages that the job can use.

The value set by this qualifier overrides the value defined in the user authorization file (UAF) of any user submitting a job to the queue.

If you specify 0 or NONE, the working set default value defaults to the value specified in the UAF or by the SUBMIT command (if included).

You also can specify this qualifier for an output execution queue. Used in this context, the /WSDEFAULT qualifier establishes the working set default of the symbiont process for an output execution queue when the symbiont process is created.

For more information about the way a working set default affects batch jobs, see Table DCL1–2.

/WSEXTENT=n

Defines for the batch job a working set extent, the maximum amount of physical memory that the job can use. The job only uses the maximum amount of physical memory when the system has excess free pages. The value set by this qualifier overrides the value defined in the user authorization file (UAF) of any user submitting a job to the queue.

If you specify 0 or NONE, the working set extent value defaults to the value specified in the UAF or by the SUBMIT command (if included).

You also can specify this qualifier for an output execution queue. Used in this context, the /WSEXTENT qualifier establishes the working set extent of the symbiont process for an output execution queue when the symbiont process is created.

For more information about the way a working set extent affects batch jobs, see Table DCL1–2.

/WSQUOTA=n

Defines for a batch job a working set quota, the amount of physical memory that is guaranteed to the job.

The value set by this qualifier overrides the value defined in the user authorization file (UAF) of any user submitting a job to the queue. If you specify 0 or NONE, the working set quota value defaults to the value specified in the UAF or by the SUBMIT command (if included).

You also can specify this qualifier for an output execution queue. Used in this context, the /WSQUOTA qualifier establishes the working set quota of the symbiont process for an output execution queue when the symbiont process is created.

Working set default, working set quota, and working set extent values are included in each user record in the system UAF. You can specify working set values for individual jobs or for all jobs in a given queue. The decision table (Table DCL1–2) shows the action taken for different combinations of specifications that involve working set values.

Table DCL1–2 Working Set Default, Extent, and Quota Decision

Value Specified by the SUBMIT Command?	Value Specified for the Queue?	Action Taken
No	No	Use the UAF value.
No	Yes	Use value for the queue.
Yes	Yes	Use smaller of the two values.
Yes	No	Compare specified value with UAF value; use the smaller.

EXAMPLES

1
```
$ INITIALIZE/QUEUE/START/BATCH/JOB_LIMIT=3 SYS$BATCH
$ INITIALIZE/QUEUE/START/BATCH/JOB_LIMIT=1/WSEXTENT=2000 BIG_BATCH
```

In this example, the first INITIALIZE/QUEUE command creates a batch queue called SYS$BATCH that can be used for any batch job. The /JOB_LIMIT qualifier allows three jobs to execute concurrently. The second INITIALIZE/QUEUE command creates a second batch queue called BIG_BATCH that is designed for large jobs. Only one job can execute at a time. The working set extent can be as high as 2000 blocks.

2
```
$ INITIALIZE/QUEUE/START/DEFAULT=(FLAG,TRAILER=ONE) /ON=LPA0: LPA0_PRINT
$ INITIALIZE/QUEUE/START/DEFAULT=(FLAG,TRAILER=ONE)/BLOCK_LIMIT=(1000,"")-
_$ /ON=LPB0: LPB0_PRINT
$ INITIALIZE/QUEUE/START/GENERIC=(LPA0_PRINT,LPB0_PRINT) SYS$PRINT
$ INITIALIZE/QUEUE/START/FORM_MOUNTED=LETTER/BLOCK_LIMIT=50/ON=TXA5: LQP
```

In this example, the first three INITIALIZE/QUEUE commands set up printer queues. Both queue LPA0_PRINT and LPB0_PRINT are set up to put a flag page before each file within a job and a trailer page after only the last page in a job. In addition, LPB0_PRINT has a minimum block size of 1000. Thus only print jobs larger than 1000 blocks can execute on that queue. SYS$PRINT is established as a generic queue that can direct jobs to either LPA0_PRINT or LPB0_PRINT. Jobs that are too small to run on LPB0_PRINT will be queued from SYS$PRINT to LPA0_PRINT.

The last INITIALIZE/QUEUE command sets up a terminal queue on TXA5. A job queued with a form that has a stock type other than the stock type of form LETTER remains pending in the queue until a form with the same stock type is mounted on the queue, or until the entry is deleted from the queue or moved to another queue. LETTER has been established at this site to indicate special letterhead paper. The block size limit is 50, indicating that this queue is reserved for jobs smaller than 51 blocks.

INQUIRE

Reads a value from SYS$COMMAND (usually the terminal in interactive mode or the next line in the main command procedure) and assigns it to a symbol.

FORMAT

INQUIRE *symbol-name [prompt-string]*

PARAMETERS

symbol-name
Specifies a symbol consisting of 1 to 255 alphanumeric characters.

prompt-string
Specifies the prompt to be displayed at the terminal when the INQUIRE command is executed. String values are automatically converted to uppercase. Also, any leading and trailing spaces and tabs are removed, and multiple spaces and tabs between characters are compressed to a single space.

Enclose the prompt in quotation marks (" ") if it contains lowercase characters, punctuation, multiple blanks or tabs, or an at sign (@). To denote an actual quotation mark in a prompt-string, enclose the entire string in quotation marks and use two consecutive quotation marks ("") within the string.

When the system displays the prompt string at the terminal, it generally places a colon (:) and a space at the end of the string. (See the /PUNCTUATION qualifier.)

If you do not specify a prompt string, the command interpreter uses the symbol name to prompt for a value.

DESCRIPTION

The INQUIRE command displays the prompting message to and reads the response from the input stream established when your process was created. This means that when the INQUIRE command is executed in a command procedure executed interactively, the prompting message is always displayed on the terminal, regardless of the level of nesting of command procedures. Note that input to the INQUIRE command in command procedures will be placed in the RECALL buffer.

When you enter a response to the prompt string, the value is assigned as a character string to the specified symbol. Lowercase characters are automatically converted to uppercase, leading and trailing spaces and tabs are removed, and multiple spaces and tabs between characters are compressed to a single space. To prohibit conversion to uppercase and retain space and tab characters, place quotation marks around the string.

To use symbols or lexical functions when you enter a response to the prompt string, use apostrophes (') to request symbol substitution.

Note that you can also use the READ command to obtain data interactively from the terminal. The READ command accepts data exactly as the user types it; characters are not automatically converted to uppercase and spaces are not compressed. However, symbols and lexical functions will not be translated even if you use apostrophes to request symbol substitution.

When an INQUIRE command is entered in a batch job, the command reads the response from the next line in the command procedure; if procedures are nested, it reads the response from the first level command procedure. If the next line in the batch job command procedure begins with a dollar sign ($), the line is interpreted as a command, not as a response to the INQUIRE command. The INQUIRE command then assigns a null string to the specified symbol, and the batch job continues processing with the command on the line following the INQUIRE command.

QUALIFIERS

/GLOBAL

Specifies that the symbol be placed in the global symbol table. If you do not specify the /GLOBAL qualifier, the symbol is placed in the local symbol table.

/LOCAL (default)

Specifies that the symbol be placed in the local symbol table for the current command procedure.

/PUNCTUATION (default)
/NOPUNCTUATION

Inserts a colon and a space after the prompt when it is displayed on the terminal. To suppress the colon and space, specify the /NOPUNCTUATION qualifier.

EXAMPLES

```
$ INQUIRE CHECK "Enter Y[ES] to continue"
$ IF .NOT. CHECK THEN EXIT
```

The INQUIRE command displays the following prompting message at the terminal:

```
Enter Y[ES] to continue:
```

The INQUIRE command prompts for a value, which is assigned to the symbol CHECK. The IF command tests the value assigned to the symbol CHECK. If the value assigned to CHECK is true (that is, an odd numeric value, a character string that begins with a T, t, Y, or y, or an odd numeric character string), the procedure continues executing.

If the value assigned to CHECK is false (that is, an even numeric value, a character string that begins with any letter except T, t, Y, or y, or an even numeric character string), the procedure exits.

2
```
$ INQUIRE COUNT
$ IF COUNT .GT. 10 THEN GOTO SKIP
   .
   .
   .
$ SKIP:
```

The INQUIRE command prompts for a count with the following message:

```
COUNT:
```

Then the command procedure uses the value of the symbol COUNT to determine whether to execute the next sequence of commands or to transfer control to the line labeled SKIP.

3
```
$ IF P1 .EQS. "" THEN INQUIRE P1 "FILE NAME"
$ FORTRAN 'P1'
```

The IF command checks whether a parameter was passed to the command procedure by checking if the symbol P1 is null; if it is, it means that no parameter was specified, and the INQUIRE command is issued to prompt for the parameter. If P1 was specified, the INQUIRE command is not executed, and the FORTRAN command compiles the name of the file specified as a parameter.

INSTALL

Invokes the Install Utility, which enhances the performance of selected executable and shareable images by making them "known" to the system and assigning them appropriate attributes. For a complete description of the Install Utility, see the *VMS Install Utility Manual*.

FORMAT   **INSTALL**   *[subcommand] [filespec]*

JOB

Identifies the beginning of a batch job submitted through a card reader. Each batch job submitted through the system card reader must be preceded by a JOB card.

JOB cannot be abbreviated.

FORMAT   **$ JOB** *user-name*

PARAMETER   *user-name*
Identifies the user name under which the job is to be run. Specify the user name as you would during the login procedure.

DESCRIPTION   The JOB card identifies the user submitting the job and is followed by a PASSWORD card giving the password. (Although the PASSWORD card is required, you do not have to use a password on the card if the account has a null password.)

The user name and password are validated by the system authorization file in the same manner as they are validated in the login procedure. The process that executes the batch job is assigned the disk and directory defaults and privileges associated with the user account. If a LOGIN.COM file exists for the specified user name, it is executed at the start of the job.

The end of a batch job is signaled by the EOJ command, by an EOF card (12-11-0-1-6-7-8-9 overpunch), or by another JOB card.

QUALIFIERS   */AFTER=time*
Holds the job until the specified time. If the specified time has already passed, the job is queued for immediate processing.

The time can be specified as either an absolute time or a combination of absolute and delta times. For complete information on specifying time values, see the *VMS DCL Concepts Manual*.

/CHARACTERISTICS=(characteristic[,...])
Specifies one or more characteristics required for processing the job. If you specify only one characteristic, you can omit the parentheses. Codes for characteristics are installation-defined. Use the SHOW QUEUE /CHARACTERISTICS command to see which characteristics are available on your system.

All the characteristics specified for the job must also be specified for the queue that will execute the job. If not, the job remains pending in the queue until the queue characteristics are changed or the entry is deleted with the DELETE/ENTRY command. Users need not specify every characteristic of a queue with the JOB command as long as the ones they specify are a subset of the characteristics set for that queue. The job also runs if no characteristics are specified.

/CLI=file-name

Specifies a different command language interpreter (CLI) with which to process the job. The *file-name* parameter specifies that the CLI be SYS$SYSTEM:*filename*.EXE. The default CLI is that defined in the user authorization file (UAF).

/CPUTIME=n

Specifies a CPU time limit for the batch job. Time can be specified as delta time, 0, NONE, or INFINITE. (For information on specifying time values, see the *VMS DCL Concepts Manual*.)

When you need less CPU time than authorized, use the /CPUTIME qualifier to override the base queue value established by the system manager or the value authorized in your UAF. Specify 0 or INFINITE to request an infinite amount of time. Specify NONE when you want the CPU time to default to your UAF value or the limit specified on the queue. Note that you cannot request more time than permitted by the base queue limits or your UAF.

/DELETE (default)
/NODELETE

Controls whether the batch input file is deleted after the job is processed. If you specify the /NODELETE qualifier, the file is saved in the user's default directory under the default name INPBATCH.COM. If you specify the /NAME qualifier, the file name of the batch input file is the same as the job name you supply with the /NAME qualifier.

/HOLD
/NOHOLD (default)

Controls whether or not the job is to be made available for immediate processing.

If you specify the /HOLD qualifier, the job is not released for processing until you specifically release it with the /NOHOLD or the /RELEASE qualifier of the SET QUEUE/ENTRY command.

/KEEP
/NOKEEP (default)

Controls whether the log file is deleted after it is printed. The /NOKEEP qualifier is the default unless you specify the /NOPRINTER qualifier.

/LOG_FILE=filespec
/NOLOG_FILE

Controls whether a log file with the specified name is created for the job or whether a log file is created.

When you use the /LOG_FILE qualifier, the system writes the log file to the file you specify. If you use the /NOLOG_FILE qualifier, no log file is created. If you specify neither form of the qualifier, the log file is written to a file in your default directory that has the same file name as the first command file in the job and a file type of LOG. Using neither the /LOG_FILE nor the /NOLOG_FILE qualifier is the default.

You can use the /LOG_FILE qualifier to specify that the log file be written to a different device. Logical names that occur in the file specification are translated at the time the job is submitted. The process executing the batch job must have access to the device on which the log file will reside.

If you omit the /LOG_FILE qualifier and specify the /NAME qualifier, the log file is written to a file having the same file name as that specified by the /NAME qualifier and the file type LOG.

/NAME=job-name

Specifies a string to be used as the job name and as the file name for both the batch job log file and the command file. The job name must be 1 to 39 alphanumeric characters and must be a valid file name. The default log file name is INPBATCH.LOG; the default command file name is INPBATCH.COM.

/NOTIFY
/NONOTIFY (default)

Controls whether a message is broadcast to any terminal at which you are logged in, notifying you when your job completes or aborts.

/PARAMETERS=(parameter[,...])

Specifies 1 to 8 optional parameters that can be passed to the command procedure. The parameters define values to be equated to the symbols P1 to P8 in the batch job. The symbols are local to the specified command procedure.

If you specify only one parameter, you can omit the parentheses.

The commas (,) delimit individual parameters. If the parameter contains any spaces, special characters or delimiters, or lowercase characters, enclose it in quotation marks (" "). Individual parameters cannot exceed 255 characters.

/PRINTER=queue-name
/NOPRINTER

Controls whether the job log file is queued to the specified queue for printing when the job is complete. The default print queue for the log file is SYS$PRINT.

If you specify the /NOPRINTER qualifier, the /KEEP qualifier is assumed.

/PRIORITY=n

Requires OPER (operator) or ALTPRI (alter priority) privilege to raise the priority above the value of the SYSGEN parameter MAXQUEPRI.

Specifies the job scheduling priority for the specified job. The value of n is an integer from 0 to 255, where 0 is the lowest priority and 255 is the highest.

The default value for the /PRIORITY qualifier is the value of the SYSGEN parameter DEFQUEPRI. No privilege is needed to set the priority lower than the MAXQUEPRI value.

The /PRIORITY qualifier has no effect on the process priority. The queue establishes the process priority.

/QUEUE=queue-name[:]

Specifies the name of the batch queue in which the job is to be entered. If you do not specify the /QUEUE qualifier, the job is placed in the default system batch job queue, SYS$BATCH.

/RESTART
/NORESTART (default)

Specifies whether the job restarts after a system failure or a STOP/QUEUE/REQUEUE command.

/TRAILING_BLANKS (default)
/NOTRAILING_BLANKS

Controls whether input cards in the card deck are read in card image form or input records are truncated at the last nonblank character. By default, the system does not remove trailing blanks from records read through the card reader. Use the /NOTRAILING_BLANKS qualifier to request that input records be truncated.

/WSDEFAULT=n

Defines a working set default for the batch job; the /WSDEFAULT qualifier overrides the working set size specified in the user authorization file (UAF). The value n can be any integer from 1 to 65,535, 0, or the keyword NONE.

Use this qualifier to impose a value lower than the base queue value established by the system manager or lower than the value authorized in your UAF. A value of 0 or the keyword NONE sets the default value to the value specified either in your UAF or by the working set quota established for the queue. You cannot request a value higher than your default.

/WSEXTENT=n

Defines a working set extent for the batch job; the /WSEXTENT qualifier overrides the working set extent in the UAF. The value n can be any integer from 1 to 65,535, 0, or the keyword NONE.

To impose a lower value, use this qualifier to override the base queue value established by the system manager rather than the value authorized in your UAF. A value of 0 or the keyword NONE sets the default value either to the value specified in the UAF or working set extent established for the queue. You cannot request a value higher than your default.

/WSQUOTA=n

Defines the maximum working set size (working set quota) for the batch job; the /WSQUOTA qualifier overrides the value in the UAF. The value n can be any integer from 1 to 65,535, 0, or the keyword NONE.

Use this qualifier to impose a value lower than the base queue value established by the system manager or lower than the value authorized in your UAF. Specify 0 or NONE if you want the working set quota defaulted to either your UAF value or the working set quota specified on the queue. You cannot request a value higher than your default.

EXAMPLES

1

```
$ EOJ
$ PRINT AVERAGE
...Input Data...
$ RUN AVERAGE
$ LINK AVERAGE
...Source Statements...
$ FORTRAN SYS$INPUT:AVERAGE
$ ON WARNING THEN EXIT
$ PASSWORD HENRY
$ JOB HIGGINS
```
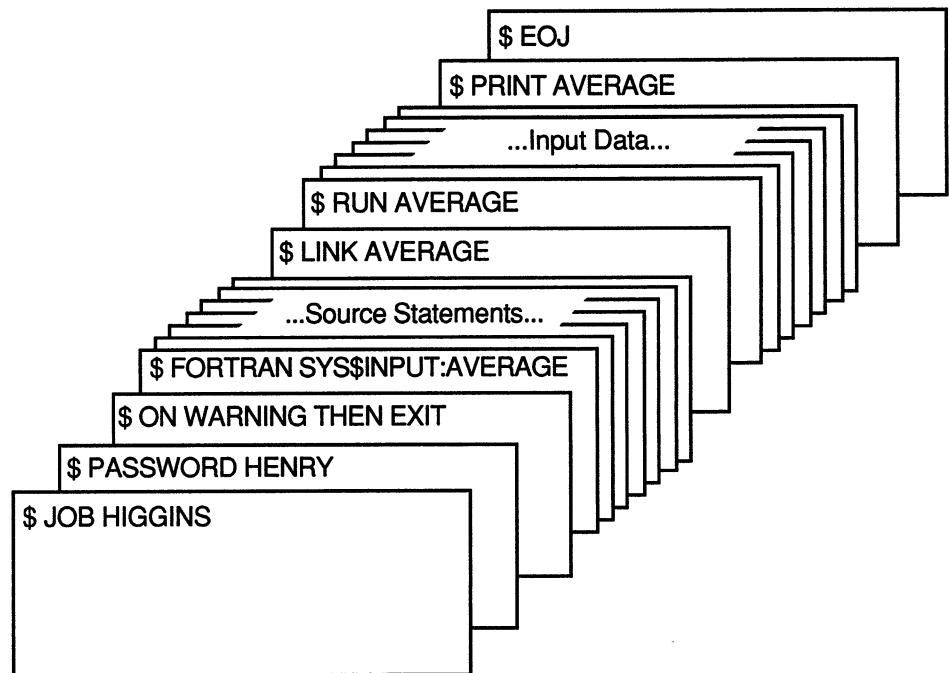
ZK-0787-GE

The JOB and PASSWORD cards identify and authorize the user HIGGINS to enter batch jobs. The command stream consists of a FORTRAN command and FORTRAN source statements to be compiled. The file name AVERAGE following the device name SYS$INPUT provides the compiler with a file name for the object and listing files. The output files are cataloged in user HIGGINS's default directory.

If the compilation is successful, the LINK command creates an executable image and the RUN command executes it. Input for the program follows the RUN command in the command stream. The last command in the job prints the program listing. The last card in the deck contains the EOJ (end of job) command.

2
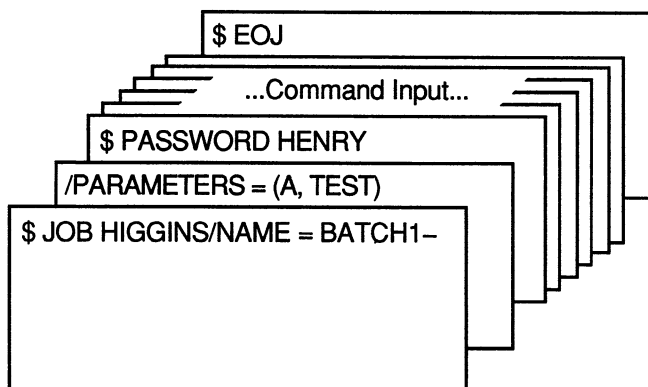
ZK–0788–GE

The /NAME qualifier on the JOB card specifies a name for the batch job. When the job completes, the printed log file is identified as BATCH1.LOG. The JOB command is continued onto a second card with the continuation character (-). The /PARAMETERS qualifier defines P1 as A and P2 as TEST. The last card in the deck contains the EOJ (end of job) command.

Lexical Functions

A set of functions that return information about character strings and attributes of the current process.

DESCRIPTION

The command language includes constructs, called lexical functions, that return information about the current process and about arithmetic and string expressions. The functions are called lexical functions because the command interpreter evaluates them during the command input scanning (or lexical processing) phase of command processing.

You can use lexical functions in any context in which you normally use symbols or expressions. In command procedures, you can use lexical functions to translate logical names, to perform character string manipulations, and to determine the current processing mode of the procedure.

The general format of a lexical function is as follows:

F$function-name([args,...])

where:

F$	Indicates that what follows is a lexical function.
function-name	A keyword specifying the function to be evaluated. Function names can be truncated to any unique abbreviation.
()	Enclose function arguments, if any. The parentheses are required for all functions, including functions that do not accept any arguments.
args,...	Specify arguments for the function, if any, using integer or character string expressions.

For more information on specifying expressions, see the *VMS DCL Concepts Manual*.

Table DCL1–3 lists each lexical function and briefly describes the information that each function returns. A detailed description of each function, including examples, is given in the following pages.

Table DCL1–3 Summary of Lexical Functions

Function	Description
F$CONTEXT	Specifies selection criteria for use with the F$PID function.
F$CSID	Returns a cluster identification number and updates the context symbol to point to the current position in the system's cluster node list.
F$CVSI	Extracts bit fields from character string data and converts the result, as a signed value, to an integer.

Table DCL1–3 (Cont.)   Summary of Lexical Functions

Function	Description
F$CVTIME	Retrieves information about an absolute, combination, or delta time string.
F$CVUI	Extracts bit fields from character string data and converts the result, as an unsigned value, to an integer.
F$DEVICE	Returns device names of all devices on a system that meet the specified selection criteria.
F$DIRECTORY	Returns the current default directory name string.
F$EDIT	Edits a character string based on the edits specified.
F$ELEMENT	Extracts an element from a string in which the elements are separated by a specified delimiter.
F$ENVIRONMENT	Obtains information about the DCL command environment.
F$EXTRACT	Extracts a substring from a character string expression.
F$FAO	Invokes the $FAO system service to convert the specified control string to a formatted ASCII output string.
F$FILE_ATTRIBUTES	Returns attribute information for a specified file.
F$GETDVI	Invokes the $GETDVI system service to return a specified item of information for a specified device.
F$GETJPI	Invokes the $GETJPI system service to return accounting, status, and identification information for a process.
F$GETQUI	Invokes the $GETQUI system service to return information about queues, batch and print jobs currently in those queues, form definitions, and characteristic definitions kept in the system job queue file.
F$GETSYI	Invokes the $GETSYI system service to return status and identification information about the local system, or about a node in the local cluster, if your system is part of a cluster.
F$IDENTIFIER	Converts an identifier in named format to its integer equivalent, or vice versa.
F$INTEGER	Returns the integer equivalent of the result of the specified expression.
F$LENGTH	Returns the length of a specified string.
F$LOCATE	Locates a character or character substring within a string and returns its offset within the string.
F$MESSAGE	Returns the message text associated with a specified system status code value.

Lexical Functions

Table DCL1–3 (Cont.) Summary of Lexical Functions

Function	Description
F$MODE	Shows the mode in which a process is executing.
F$PARSE	Invokes the $PARSE RMS service to parse a file specification and return either the expanded file specification or the particular file specification field that you request.
F$PID	For each invocation, returns the next process identification number in sequence.
F$PRIVILEGE	Returns a value of "TRUE" or "FALSE" depending on whether your current process privileges match the privileges listed in the argument.
F$PROCESS	Returns the current process name string.
F$SEARCH	Invokes the $SEARCH RMS service to search a directory file, and returns the full file specification for a file you name.
F$SETPRV	Sets the specified privileges and returns a list of keywords indicating the previous state of these privileges for the current process.
F$STRING	Returns the string equivalent of the result of the specified expression.
F$TIME	Returns the current date and time of day, in the format dd-mmm-yyyy hh:mm:ss.cc.
F$TRNLNM	Translates a logical name and returns the equivalence name string or the requested attributes of the logical name.
F$TYPE	Determines the data type of a symbol.
F$USER	Returns the current user identification code (UIC).
F$VERIFY	Returns the integer 1 if command procedure verification is set on; returns the integer 0 if command procedure verification is set off. The F$VERIFY function also can set new verification states.

F$CONTEXT

Specifies selection criteria for use with the F$PID function. The F$CONTEXT function enables the F$PID function to obtain information about processes from any node in a VAXcluster.

FORMAT

F$CONTEXT(context-type, context-symbol, selection-item, selection-value, value-qualifier)

return value

A null string ("").

ARGUMENTS

context-type

Specifies the type of context to be built. At present, the only context type available is PROCESS, which is used in constructing selection criteria for F$PID.

context-symbol

Specifies a symbol that DCL uses to refer to the context memory being constructed by the F$CONTEXT function. The function F$PID uses this context symbol to process the appropriate list of process identification (PID) numbers.

Specify the context symbol by using a symbol. The first time you use the F$CONTEXT function in a command procedure, use a symbol that is either undefined or equated to the null string. The symbol created will be a local symbol of type "PROCESS_CONTEXT". When the context is no longer valid—that is, when all PIDs have been retrieved by calls to the F$PID function or an error occurs during one of these calls—the symbol no longer has a type of "PROCESS_CONTEXT". Then you can use the F$TYPE function in the command procedure to find out if it is necessary to cancel the context.

After setting up the selection criteria, use this context symbol when calling F$PID.

selection-item

Specifies a keyword that tells F$CONTEXT which selection criteria to use. Use only one selection-item keyword per call to F$CONTEXT.

The following table shows valid selection-item keywords for the PROCESS context type:

Selection Item	Selection Value	Value Qualifiers	Comments
ACCOUNT	String	EQL, NEQ	Valid account name or list of names. Wildcard characters (* and %) are allowed.
AUTHPRI	Integer	GEQ, GTR, LEQ, LSS, EQL, NEQ	Valid authorized base priority (0–31).
CANCEL			Cancels the selection criteria for this context.
CURPRIV	Keyword	ALL, ANY, EQL, NEQ	Valid privilege name keyword or list of keywords. For more information, see the *VMS DCL Concepts Manual*.
GRP	String	GEQ, GTR, LEQ, LSS, EQL, NEQ	Group number or name.
HW_MODEL	Integer	EQL, NEQ	Valid hardware model number.
HW_NAME	String	EQL, NEQ	Valid hardware name or a list of keywords. Wildcard characters (* and %) are allowed.
JOBPRCCNT	Integer	GEQ, GTR, LEQ, LSS, EQL, NEQ	Subprocess count for entire job.
JOBTYPE	Keyword	EQL, NEQ	Valid job-type keyword. Valid keywords are DETACHED, NETWORK, BATCH, LOCAL, DIALUP, and REMOTE. For more information, see the *VMS DCL Concepts Manual*.
MASTER_PID	String	EQL, NEQ	PID of master process.
MEM	String or Integer	GEQ, GTR, LEQ, LSS, EQL, NEQ	UIC member number or name.
MODE	Keyword	EQL, NEQ	Valid process mode keyword. Valid keywords are OTHER, NETWORK, BATCH, and INTERACTIVE. For more information, see the *VMS DCL Concepts Manual*.
NODE_CSID	Integer	EQL, NEQ	Node's cluster ID number.
NODENAME	String	EQL, NEQ	Node name or list of node names. Wildcard characters are allowed. The default is your local node. To request all nodes, use the value "*".
OWNER	String	EQL, NEQ	PID of immediate parent process.
PRCCNT	Integer	GEQ, GTR, LEQ, LSS, EQL, NEQ	Subprocess count of process.
PRCNAM	String	EQL, NEQ	Process name or list of process names. Wildcard characters are allowed.

Selection Item	Selection Value	Value Qualifiers	Comments
PRI	Integer	GEQ, GTR, LEQ, LSS, EQL, NEQ	Process priority level number (0–31).
PRIB	Integer	GEQ, GTR, LEQ, LSS, EQL, NEQ	Base process priority level number (0–31).
STATE	Keyword	EQL, NEQ	Valid process state keyword. For more information, see the description of the $GETJPI service in the *VMS System Services Reference Manual*.
STS	Keyword	EQL, NEQ	Valid process status keyword. For more information, see the description of the $GETJPI service in the *VMS System Services Reference Manual*.
TERMINAL	String	EQL, NEQ	Terminal name or list of names. Wildcard characters are allowed.
UIC	String	EQL, NEQ	User identification code (UIC) identifier (that is, of the form "[group,member]").
USERNAME	String	EQL, NEQ	User name or list of user names. Wildcard characters (* and %) are allowed.

selection-value

Specifies the value of the selection criteria. For example, to process all the processes running on node MYVAX, specify "MYVAX" with the "NODENAME" keyword. For example:

```
$  X = F$CONTEXT("PROCESS", ctx, "NODENAME", "MYVAX", "EQL")
```

Values that are lists are valid with some selection items. If you specify more than one item, separate them with commas (,). The following example specifies a list of the nodes MYVAX, HERVAX, and HISVAX:

```
$  X = F$CONTEXT("PROCESS", ctx, "NODENAME", "MYVAX,HERVAX,HISVAX", "EQL")
```

You can use wildcard characters (* and %) for some values. Using wildcard characters for selection items is similar to using wildcard characters for file names.

value-qualifier

Specifies qualifiers for selection values. You must qualify selection values.

You can qualify a number, for example, by requesting that the selection be based on the process value less than (LSS), less than or equal to (LEQ), greater than (GTR), greater than or equal to (GEQ), equal to (EQL), or not equal to (NEQ) the value specified in the call to F$PID.

You can qualify some lists with the ALL, ANY, EQL, or NEQ keywords. Such lists are usually masks such as the process privilege mask, which consists of the set of enabled privileges. ALL requires that all items in the list be true for a process; ANY requests that any item in the list be part of the attributes of a process; EQL means that the values must match

exactly (that is, values not specified must not be true of the process); and NEQ requires that the value must not match.

The difference between ALL and EQL is that the values specified with ALL must exist, but other unspecified values can exist also. EQL requires that all values specified must exist, and all others may not. For example, to request those processes whose current privileges include TMPMBX (temporary mailbox) and OPER (operator), but may include other privileges, specify the ALL keyword. To request those processes whose current privileges are TMPMBX and OPER exclusively, specify the EQL keyword.

DESCRIPTION   Use the F$CONTEXT function to set up selection criteria for the F$PID function.

The F$CONTEXT function is called as many times as necessary to produce the criteria needed; however, each call can specify only one selection item. Lists of item values are allowed, where appropriate, and more than one context can be operated upon at a time.

After establishing the selection criteria with appropriate calls to F$CONTEXT, F$PID is called repeatedly to return all the process identification (PID) numbers that meet the criteria specified in the F$CONTEXT function. When there are no more such processes, the F$PID function returns a null string.

After the F$PID function is called, the context symbol is considered "frozen"; F$CONTEXT cannot be called again with the same context symbol until the associated context selection criteria have been deleted. If you attempt to set up additional selection criteria with the same context symbol, an error message is displayed. However, the context and selection criteria are not affected and calls to the F$PID function can continue.

The F$CONTEXT function uses process memory to store the selection criteria. This memory is deleted under two circumstances. Memory is deleted when the F$PID function is called and a null string ("") is returned—that is, when all processes that meet the selection criteria have been returned. Memory also is deleted if the CANCEL selection-item keyword is used in a call to F$CONTEXT with an established context. This type of call is appropriate for a CTRL/Y operation or another condition handling routine.

EXAMPLE

```
$!Establish an error and CTRL/Y handler
$!
$ ON ERROR THEN GOTO error
$ ON CONTROL_Y THEN GOTO error
$!
$ ctx = ""
$ temp = F$CONTEXT ("PROCESS", ctx, "NODENAME", "*","EQL")
$ temp = F$CONTEXT ("PROCESS", ctx, "USERNAME", "M*,SYSTEM","EQL")
$ temp = F$CONTEXT ("PROCESS", ctx, "CURPRIV", "SYSPRV,OPER", "ALL")
```

```
$!
$!Loop over all processes that meet the selection criteria.
$!Print the PID and the name of the image for each process.
$!
$loop:
$ pid = F$PID(ctx)
$ IF pid .EQS. ""
$ THEN
$     GOTO endloop
$ ELSE
$     image = F$GETJPI(pid,"IMAGNAME")
$     SHOW SYMBOL pid
$     WRITE SYS$OUTPUT image
$     GOTO loop
$ ENDIF
$!The loop over the processes has ended.
$!
$endloop:
$!
$ EXIT
$!
$!Error handler. Clean up the context's memory with
$!the CANCEL selection item keyword.
$!
$error:
$ IF F$TYPE(ctx) .eqs. "PROCESS_CONTEXT" THEN -
-$ temp = F$CONTEXT ("PROCESS", ctx, "CANCEL")
$!
$ EXIT
```

In this example, F$CONTEXT is called three times to set up selection criteria. The first call requests that the search take place on all nodes in the cluster. The second call requests that only the processes whose user name either starts with an "M" or is "SYSTEM" be processed. The third call restricts the selection to those processes whose current privileges include both SYSPRV (system privilege) and OPER (operator) and can have other privileges set.

The command lines between the labels "loop" and "endloop" continually call F$PID to obtain the processes that meet the criteria set up in the F$CONTEXT calls. After retrieving each PID, F$GETJPI is called to return the name of the image running in the process. Finally, the procedure displays the name of the image.

In case of error or a CTRL/Y operation, control is passed to *error* and the context is closed if necessary. In this example, note the check for the symbol type PROCESS_CONTEXT. If the symbol has this type, selection criteria must be canceled by a call to F$CONTEXT. If the symbol is not of the type PROCESS_CONTEXT, either selection criteria have not been set up yet in F$CONTEXT, or the symbol was used with F$PID until an error occurred or until the end of the process list was reached.

F$CSID

Returns a cluster identification number and updates the context symbol to point to the current position in the system's cluster node list.

FORMAT   **F$CSID**(context-symbol)

---

return value   A character string containing the system cluster identification number in the system's list of clustered nodes. If the current system is not a member of a cluster, the first return value is null.

After the last system name is returned, the F$CSID function returns a null string ("").

ARGUMENTS   *context-symbol*

Specifies a symbol that DCL uses to store a pointer into the system's list of clustered nodes. The F$CSID function uses this pointer to return a cluster node name.

Specify the **context-symbol** argument by using a symbol. The first time you use the F$CSID function, use a symbol that is either undefined or equated to the null string.

If the **context-symbol** argument is undefined or equated to a null string, the F$CSID function returns the first system in the system's cluster node list. Subsequent calls to the F$CSID function will return the rest of the nodes in the cluster.

DESCRIPTION   The F$CSID function returns a cluster identification number, and updates the context symbol to point to the current position in the system's cluster node list.

If the current system is not a member of a cluster, the first return value is null.

You can use the F$CSID function to obtain all of the cluster identification numbers on the system. For each cluster identification returned, the F$GETSYI function can be used to obtain information about the particular system.

Once the **context-symbol** argument is initialized by the first call, each subsequent F$CSID function call returns another node in the cluster. (Note that the cluster identification numbers are returned in random order.) After the last cluster system in the list is returned, the F$CSID function returns a null string.

EXAMPLE

```
$ IF F$GETSYI("CLUSTER_MEMBER") .EQS. "FALSE" THEN GOTO NOT_CLUSTER
$ CONTEXT = ""
$START:
$   id = F$CSID (CONTEXT)
$   IF id .EQS. "" THEN EXIT
$   nodename = F$GETSYI ("NODENAME",,id)
$   WRITE SYS$OUTPUT nodename
$   GOTO start
$NOT_CLUSTER:
$ WRITE SYS$OUTPUT "Not a member of a cluster."
$ EXIT
```

This command procedure uses the F$CSID function to display a list of cluster system names. The assignment statement declares the symbol CONTEXT, which is used as the **context-symbol** argument for the F$CSID function. Because CONTEXT is equated to a null string, the F$CSID function will return the first cluster identification number in the cluster node list.

If the F$CSID function returns a null value, then the command procedure either is at the end of the list, or is attempting this operation on a non-clustered node. The call to F$GETSYI checks whether the current node is a member of a cluster. The command procedure will exit on this condition.

If the F$CSID function does not return a null value, then the command procedure uses the identification number as the third argument to the F$GETSYI function to obtain the name of the system. The name is then displayed using the WRITE command.

F$CVSI

Converts the specified bits in the specified character string to a signed number.

FORMAT   **F$CVSI**(start-bit,number-of-bits,string)

---

return value   The integer equivalent of the extracted bit field, converted as a signed value.

ARGUMENTS

start-bit
Specifies the offset of the first bit to be extracted. The low-order (rightmost) bit of a string is position number 0 for determining the offset. Specify the offset as an integer expression.

If you specify an expression with a negative value, or with a value that exceeds the number of bits in the string, then DCL displays the INVRANGE error message.

number-of-bits
Specifies the length of the bit string to be extracted, which must be less than or equal to the number of bits in the string.

If you specify an expression with a negative value, or with a value that is invalid when added to the bit position offset, then DCL displays the INVRANGE error message.

string
Specifies the string from which the bits are taken. Specify the string as a character string expression.

EXAMPLES

1
```
$ A[0,32] = %X2B
$ SHOW SYMBOL A
  A = "+..."
$ X = F$CVSI(0,4,A)
$ SHOW SYMBOL X
  X = -5    Hex = FFFFFFFB   Octal = 37777777773
```

This example uses an arithmetic overlay to assign the hexadecimal value 2B to all 32 bits of the symbol A. For more information on arithmetic overlays, see the description of the assignment statement (=).

The symbol A has a string value after the overlay because it was previously undefined. (If a symbol is undefined, it has a string value as a result of an arithmetic overlay. If a symbol was previously defined, it retains the same data type after the overlay.) The hexadecimal value 2B corresponds to the ASCII value of the plus sign (+).

Next, the F$CVSI function extracts the low-order 4 bits from the symbol A; the low-order 4 bits contain the binary representation of the hexadecimal value B. These bits are converted, as a signed value, to an integer. The converted value, –5, is assigned to the symbol X.

2
```
$  SYM[0,32] = %X2A
$  SHOW SYMBOL SYM
   SYM = "*..."
$  Y = F$CVSI(0,33,SYM)
%DCL-W-INVRANGE, field specification is out of bounds - check sign and size
$  SHOW SYMBOL Y
%DCL-W-UNDSYM, undefined symbol - check spelling
```

In this example, the width argument specified with the F$CVSI function is too large. Therefore, DCL issues an error message and the symbol Y is not assigned a value.

F$CVTIME

Converts an absolute or a combination time string to a string of the form *yyyy-mm-dd hh:mm:ss.cc*. The F$CVTIME function can also return information about an absolute, combination, or delta time string.

FORMAT

F$CVTIME(*[input_time]* *[,output_time_format]* *[,output_field]*)

return value

A character string containing the requested information.

ARGUMENTS

input_time

Specifies a string containing an absolute, a delta, or a combination time, or TODAY, TOMORROW, or YESTERDAY. Specify the input time string as a character string expression. If the **input_time** argument is omitted or is specified as a null string (""), the current system date and time, in absolute format, is used. If parts of the date field are omitted, the missing values default to the current date. If parts of the time field are omitted, the missing values default to zero.

For more information on these time formats, see the *VMS DCL Concepts Manual*.

If the **input_time** argument is a delta time, you must specify the **output_time_format** argument as DELTA.

output_time_format

Specifies the time format for the information you want returned. Specify the **output_time_format** argument as one of the following character string expressions:

ABSOLUTE	The requested information should be returned in absolute time format, which is *dd-mmm-yyyy hh:mm:ss.cc*.
COMPARISON (default)	The requested information should be returned in the form *yyyy-mm-dd hh:mm:ss.cc* (used for comparing two times).
DELTA	The requested information should be returned in delta format, which is *dddd-hh:mm:ss.cc*. If you specify DELTA as the **output_time_format** argument, then you must also provide a delta time specification for the **input_time** argument.

output_field

Specifies a character string expression containing one of the following (do not abbreviate): DATE, MONTH, DATETIME (default), SECOND, DAY, TIME, HOUR, WEEKDAY, HUNDREDTH, YEAR, MINUTE. The information is returned in the time format specified by the **output_time_format** argument. If the **input_time** argument is a delta time and the **output_time_format** argument is DELTA, you cannot specify MONTH, WEEKDAY, or YEAR.

When the weekday is returned, the first letter is in uppercase, and the following letters are in lowercase.

DESCRIPTION When using the F$CVTIME function, you can omit optional arguments that can be used to the right of the last argument you specify. However, you must include commas (,) as placeholders if you omit optional arguments to the left of the last argument you specify.

When specifying the input time argument in either absolute or combination time format, you can specify ABSOLUTE or COMPARISON as the **output_time_format** argument; you cannot specify DELTA.

When specifying the **input_time** argument in delta time format, you must specify DELTA as the **output_time_format** argument.

EXAMPLES

1
```
$  TIME = F$TIME()
$  SHOW SYMBOL TIME
   TIME = "19-APR-1990 10:56:23.10"
$  TIME = F$CVTIME(TIME)
$  SHOW SYMBOL TIME
   TIME = "1990-04-19 10:56:23.10"
```

This example uses the F$TIME function to return the system time as a character string and to assign the time to the symbol TIME. Then the F$CVTIME function is used to convert the system time to an alternate time format. Note that you do not need to place quotation marks (" ") around the argument TIME because it is a symbol. Symbols are automatically evaluated when they are used as arguments for lexical functions.

You can use the resultant string to compare two dates (using .LTS. and .GTS. operators). For example, you can use F$CVTIME to convert two time strings and store the results in the symbols TIME_1 and TIME_2. You can compare the two values, and branch to a label, based on the following results:

```
$ IF TIME_1 .LTS. TIME_2 THEN GOTO FIRST
```

2
```
$  NEXT = F$CVTIME("TOMORROW",,"WEEKDAY")
$  SHOW SYMBOL NEXT
   NEXT = "Tuesday"
```

In this example, the F$CVTIME returns the weekday that corresponds to the absolute time keyword "TOMORROW". You must enclose the arguments "TOMORROW" and "WEEKDAY" in quotation marks because they are character string expressions. Also, you must include a comma as a placeholder for the **output_time_format** argument that is omitted.

F$CVUI

Extracts bit fields from character string data and converts the result to an unsigned number.

FORMAT   **F$CVUI**(start-bit,number-of-bits,string)

---

return value   The integer equivalent of the extracted bit field, converted as an unsigned value.

ARGUMENTS   **start-bit**
Specifies the offset of the first bit to be extracted. The low-order (rightmost) bit of a string is position number 0 for determining the offset. Specify the offset as an integer expression.

If you specify an expression with a negative value, or with a value that exceeds the number of bits in the string, DCL displays the INVRANGE error message.

number-of-bits
Specifies the length of the bit string to be extracted, which must be less than or equal to the number of bits in the string argument.

If you specify an expression with a negative value, or with a value that is invalid when added to the bit position offset, DCL displays the INVRANGE error message.

string
Specifies the character string to be edited.

EXAMPLE

```
$  A[0,32] = %X2B
$  SHOW SYMBOL A
  A = "+..."
$  X = F$CVUI(0,4,A)
$  SHOW SYMBOL X
  X = 11    Hex = 0000000B   Octal = 00000000013
```

This example uses an arithmetic overlay to assign the hexadecimal value 2B to all 32 bits of the symbol A. The symbol A has a string value after the overlay because it was previously undefined. (If a symbol is undefined, it has a string value as a result of an arithmetic overlay. If a symbol was previously defined, it retains the same data type after the overlay.) The hexadecimal value 2B corresponds to the ASCII character "+".

Next, the F$CVUI function extracts the low-order 4 bits from the symbol A; the low-order 4 bits contain the binary representation of the hexadecimal value B. These bits are converted, as a signed value, to an integer. The converted value, 11, is assigned to the symbol X.

F$DEVICE

Returns the device names of all devices on a system that meet the specified selection criteria.

Note that the device names are returned in random order.

FORMAT

F$DEVICE([search_devnam],[devclass],[devtype], [stream-id])

return value

A character string containing the name of a device in the system's list of devices.

After the last device name in the system's device list is returned, the F$DEVICE function returns a null string ("").

ARGUMENTS

search_devnam
Specifies the name of the device for which F$DEVICE is to search. Wildcard characters (* and %) are allowed in the **search_devnam** argument.

Specify the **search_devnam** argument as a character string expression.

devclass
Specifies the device class for which F$DEVICE is to search. Specify the **devclass** argument as a character string expression that corresponds to a valid device class name. See Table DCL1–7 under F$GETDVI.

devtype
Specifies the device type for which F$DEVICE is to search. Specify the **devtype** argument as a character string expression that corresponds to a valid type name. Valid type names can be found in Table DCL1–8 under F$GETDVI.

Note: **Specifying a device type without specifying a device class will result in an error.**

stream-id
A positive integer representing the search stream identification number.

The search stream identification number is used to maintain separate search contexts when you use the F$DEVICE function more than once and when you supply different search criteria. If you use the F$DEVICE function more than once in a command procedure and if you also use different search criteria, specify **stream-id** arguments to identify each search separately.

If the search criteria are changed in a call before the device name list is exhausted, the context will be reinitialized and the search will restart.

If you omit the **stream-id** argument, the F$DEVICE function assumes an implicit single search stream. That is, the F$DEVICE function starts searching at the beginning each time you specify different search criteria.

DESCRIPTION

The F$DEVICE function allows you to search for devices that meet certain search criteria by using the $DEVICE_SCAN system service.

The F$DEVICE function allows wildcard searches based on the device name, the device class, or the device type.

You can use the F$DEVICE function in a loop in a command procedure to return device names that match the specified selection criteria. Each time the F$DEVICE function is executed, it returns the next device on the system that matches the selection criteria. Note that devices are returned in no particular order. After the last device name is returned, the next F$DEVICE function returns a null string.

Note that you must maintain the context of the search string explicitly (by specifying the **stream-id** argument) or implicitly (by omitting the **stream-id** argument). In either case, you must specify the same selection criteria each time you execute the F$DEVICE system service with the same (explicit or implicit) stream.

EXAMPLE

```
$ START:
$      DEVICE_NAME = F$DEVICE("*0:","DISK","RA60")
$      IF DEVICE_NAME .EQS. "" THEN EXIT
$      SHOW SYMBOL DEVICE_NAME
$      GOTO START
```

This command procedure displays the device names of all the RA60s on a unit numbered 0.

Because no **stream-id** argument is specified, F$DEVICE uses an implicit search stream. Each subsequent use of the F$DEVICE function uses the same search criteria to return the next device name. After the last device name is displayed, the F$DEVICE function returns a null string and the procedure exits.

F$DIRECTORY

Returns the current default directory name string. The F$DIRECTORY function has no arguments, but must be followed by parentheses.

FORMAT   F$DIRECTORY()

return value   A character string for the current default directory name, including brackets ([ ]). If you use the SET DEFAULT command and specify angle brackets (<>) in a directory specification, the F$DIRECTORY function returns angle brackets in the directory string.

ARGUMENTS   *None.*

DESCRIPTION   You can use the F$DIRECTORY function to save the name of the current default directory in a command procedure, to change the default to another directory to do work, and to later restore the original setting.

EXAMPLE

```
$   SAVE_DIR = F$DIRECTORY()
$   SET DEFAULT [MALCOLM.TESTFILES]
    .
    .
    .
$   SET DEFAULT 'SAVE_DIR'
```

This example shows an excerpt from a command procedure that uses the F$DIRECTORY function to save the current default directory setting. The assignment statement equates the symbol SAVE_DIR to the current directory. Then the SET DEFAULT command establishes a new default directory. Later, the symbol SAVE_DIR is used in the SET DEFAULT command that restores the original default directory.

Note that you can use the F$ENVIRONMENT function with the DEFAULT keyword to return the default disk and directory. You should use the F$ENVIRONMENT function rather than the F$DIRECTORY function in situations involving more than one disk.

F$EDIT

Edits the character string based on the edits specified in the **edit-list**
argument.

FORMAT   **F$EDIT**(string, edit-list)

---

return value   A character string containing the specified edits.

ARGUMENTS   *string*
Specifies a character string to be edited. Quoted sections of the string are
not edited.

edit-list
Specifies a character string containing one or more of the following
keywords which specify the types of edits to be made to the string:

Edit	Action
COLLAPSE	Removes all spaces or tabs.
COMPRESS	Replaces multiple spaces or tabs with a single space.
LOWERCASE	Changes all uppercase characters to lowercase.
TRIM	Removes leading and trailing spaces or tabs.
UNCOMMENT	Removes comments.
UPCASE	Changes all lowercase characters to uppercase.

If you specify more than one keyword, separate them with commas (,). Do
not abbreviate these keywords.

Edits are not applied to quoted sections of strings. Therefore, if a string
contains quotation marks (" "), the characters within the quotation marks
are not affected by the edits specified in the edit list.

EXAMPLES

1   ```
$ LINE = "   THIS   LINE   CONTAINS A "" QUOTED "" WORD"
$ SHOW SYMBOL LINE
  LINE = "   THIS   LINE   CONTAINS A " QUOTED " WORD"
$ NEW_LINE = F$EDIT(LINE, "COMPRESS, TRIM")
$ SHOW SYMBOL NEW_LINE
  NEW_LINE = "THIS LINE CONTAINS A " QUOTED " WORD"
```

This example uses the F$EDIT function to compress and trim a string by
replacing multiple blanks with a single blank, and by removing leading
and trailing blanks. The string LINE contains quotation marks around
the word QUOTED. (To enter quotation marks into a character string, use
double quotation marks in the assignment statement.)

Note that the F$EDIT function does not compress the spaces in the quoted section of the string; therefore, the spaces are retained around the word QUOTED.

```
$ LOOP:
$       READ/END_OF_FILE = DONE INPUT_FILE RECORD
$       RECORD = F$EDIT(RECORD, "TRIM, UPCASE")
$       WRITE OUTPUT_FILE RECORD
$       GOTO LOOP
        .
        .
        .
```

This example sets up a loop to read records from a file, to edit them, and to write them to an output file. The edited records have leading and trailing blanks removed, and are converted to uppercase.

F$ELEMENT

Extracts one element from a string of elements.

FORMAT

F$ELEMENT*(element-number, delimiter, string)*

---

return value

A character string containing the specified element.

ARGUMENTS

element-number
Specifies the number of the element to extract (numbering begins with zero). Specify the **element-number** argument as an integer expression. If the **element-number** argument exceeds the number of elements in the string, F$ELEMENT returns the delimiter.

delimiter
Specifies a character used to separate the elements in the string. Specify the delimiter as a character string expression.

string
Specifies a string containing a delimited list of elements. Specify the string as a character string expression.

EXAMPLES

❶
```
$ DAY_LIST = "MON/TUE/WED/THU/FRI/SAT/SUN"
$ INQUIRE DAY "ENTER DAY (MON TUE WED THU FRI SAT SUN)"
$ NUM = 0
$ LOOP:
$       LABEL = F$ELEMENT(NUM,"/",DAY_LIST)
$       IF LABEL .EQS. "/" THEN GOTO END
$       IF DAY .EQS. LABEL THEN GOTO 'LABEL'
$       NUM = NUM +1
$       GOTO LOOP
$
$ MON:
        .
        .
        .
```

This example sets up a loop to test an input value against the elements in a list of values. If the value for DAY matches one of the elements in DAY_LIST, control is passed to the corresponding label. If the value returned by the F$ELEMENT function matches the delimiter, the value DAY was not present in the DAY_LIST, and control is passed to the label END.

```
2   $ ! INDEX.COM
    $ !
    $ CHAPTERS = "0,1,2,3,4,5,6,A,B,C"
    $ NEXT = 0
    $ LOOP:
    $   NEXT = NEXT + 1
    $   NUM = F$ELEMENT(NEXT,",",CHAPTERS)
    $   IF (NUM .NES. ",")
    $   THEN
    $       RUN INDEX CHAP'NUM'
    $       GOTO LOOP
    $   ENDIF
```

This example processes files named CHAP1, CHAP2, ... CHAP6, CHAPA, CHAPB, and CHAPC, in that order. (Zero is included in the CHAPTERS string to initialize the procedure logic.) NEXT is initialized to zero. The procedure enters the loop. In the first iteration, NEXT is incremented to 1 and the result of the F$ELEMENT call is the string "1". The procedure runs the index, chapter 1. In the second iteration, NEXT is incremented to 2 and the result of the F$ELEMENT call is the string "2". The procedure runs the index, chapter 2. Processing continues until the result of the F$ELEMENT call is the delimiter specified in the call.

F$ENVIRONMENT

Returns information about the current DCL command environment.

FORMAT	**F$ENVIRONMENT**(item)

return value	Information that corresponds to the specified item. The return value can be either an integer or a character string, depending on the specified item.

ARGUMENT

item
Specifies the type of information to be returned. Specify one of the following keywords (do not abbreviate these keywords):

Item	Data Type	Information Returned
CAPTIVE	String	TRUE if you are logged in to a captive account. The system manager can define captive accounts in the user authorization file (UAF) by using the Authorize Utility (AUTHORIZE).
CONTROL	String	Control characters currently enabled with SET CONTROL. Multiple characters are separated by commas; if no control characters are enabled, the null string (" ") is returned.
DEFAULT	String	Current default device and directory name. The returned string is the same as SHOW DEFAULT output.
DEPTH	Integer	Current command procedure depth. The command procedure depth is 0 when you log in interactively and when you submit a batch job. The command procedure depth is 1 when you execute a command procedure interactively or from within a batch job. A nested command procedure has a depth of 1 greater than the depth of the command procedure from which the nested procedure is executed.
DISIMAGE	String	TRUE if you are logged in to an account that does not allow the use of the RUN and MCR commands or foreign commands. The system manager can add or remove the DISIMAGE attribute for accounts in the UAF by using AUTHORIZE.
INTERACTIVE	String	TRUE if the process is executing interactively.

Item	Data Type	Information Returned
KEY_STATE	String	Current locked keypad state. See the description of the DEFINE/KEY command for more information on keypad states.
MAX_DEPTH	Integer	Maximum allowable command procedure depth.
MESSAGE	String	Current setting of SET MESSAGE qualifiers. Each qualifier in the string is prefaced by a slash (/); therefore, the output from F$ENVIRONMENT("MESSAGE") can be appended to the SET MESSAGE command to form a valid DCL command line.
NOCONTROL	String	Control characters currently disabled with SET NOCONTROL. Multiple characters are separated by commas (,); if no control characters are disabled, the null string is returned.
ON_CONTROL_Y	String	If issued from a command procedure, returns TRUE if ON_CONTROL_Y is set. ON_CONTROL_Y always returns FALSE at DCL command level.
ON_SEVERITY	String	If issued from a command procedure, returns the severity level at which the action specified with the ON command is performed. ON_SEVERITY returns NONE when SET NOON is in effect or at DCL command level.
OUTPUT_RATE	String	Delta time string containing the default output rate, which indicates how often data is written to the batch job log file while the batch job is executing. OUTPUT_RATE returns a null string if used interactively.
PROCEDURE	String	File specification of the current command procedure. PROCEDURE returns a null string if used interactively.
PROMPT	String	Current DCL prompt.
PROMPT_CONTROL	String	TRUE if a carriage return and line feed precede the prompt.
PROTECTION	String	Current default file protection. The string can be used with the SET PROTECTION /DEFAULT command to form a valid DCL command line.
RESTRICTED	String	TRUE if you are logged in to a restricted account. The system manager can define restricted accounts in the UAF by using AUTHORIZE.
SYMBOL_SCOPE	String	[NO]LOCAL, [NO]GLOBAL to indicate the current symbol scoping state.

Lexical Functions
F$ENVIRONMENT

Item	Data Type	Information Returned
VERB_SCOPE	String	[NO]LOCAL, [NO]GLOBAL to indicate the current symbol scoping state for verbs. (For more information, see the description of the SET SYMBOL command.)
VERIFY_IMAGE	String	TRUE if image verification (SET VERIFY=IMAGE) is in effect. If image verification is in effect, then the command procedure echoes input data read by images.
VERIFY_PROCEDURE	String	TRUE if procedure verification SET VERIFY=PROCEDURE is in effect. If command verification is in effect, then the command procedure echoes DCL command lines.

EXAMPLES

1
```
$  SAVE_MESSAGE = F$ENVIRONMENT("MESSAGE")
$  SET MESSAGE/NOFACILITY/NOIDENTIFICATION
     .
     .
     .
$  SET MESSAGE'SAVE_MESSAGE'
```

> This example uses the F$ENVIRONMENT function to save the current message setting before changing the setting. At the end of the command procedure, the original message setting is restored. The single quotation marks ('') surrounding the symbol SAVE_MESSAGE indicate that the value for the symbol should be substituted.

2
```
$  MAX = F$ENVIRONMENT("MAX_DEPTH")
$  SHOW SYMBOL MAX
   MAX = 32    Hex = 00000020  Octal = 00000000040
```

> This example uses the F$ENVIRONMENT function to determine the maximum depth allowable within command procedures.

3
```
$  SAVE_PROT = F$ENVIRONMENT("PROTECTION")
$  SET PROTECTION = (SYSTEM:RWED, OWNER:RWED, GROUP, WORLD)/DEFAULT
     .
     .
     .
$  SET PROTECTION = ('SAVE_PROT')/DEFAULT
```

> This example uses the F$ENVIRONMENT function to save the current default protection before changing the protection. At the end of the command procedure, the original protection is restored. You must place single quotation marks around the symbol SAVE_PROT to request symbol substitution.

F$EXTRACT

Extracts the specified characters from the specified string.

FORMAT

F$EXTRACT*(start,length,string)*

---

return value

A character string containing the characters delimited by the **start** and **length** arguments.

ARGUMENTS

start
Specifies the offset of the starting character of the string you want to extract. Specify the start argument as an integer expression that is greater than or equal to zero.

The offset is the relative position of a character or a substring with respect to the beginning of the string. Offset positions begin with zero. The string always begins with the leftmost character.

If you specify an offset that is greater than or equal to the length of the string, F$EXTRACT returns a null string ("").

length
Specifies the number of characters you want to extract; must be less than or equal to the size of the string. Specify the length as an integer expression that is greater than or equal to zero.

If you specify a length that exceeds the number of characters from the offset to the end of the string, the F$EXTRACT function returns the characters from the offset through the end of the string.

string
Specifies the character string to be edited. Specify the string as a character string expression.

EXAMPLES

1
```
$ NAME = "JOE SMITH"
$ FIRST = F$EXTRACT(0,3,NAME)
$ SHOW SYMBOL FIRST
  FIRST = "JOE"
```

This portion of a command procedure uses the F$EXTRACT function to extract the first 3 characters from the character string assigned to the symbol NAME. The offset and length arguments are integers, and the string argument is a symbol. You do not need to use quotation marks (" ") around integers or symbols when they are used as arguments for lexical functions.

2
```
$ P1 = "MYFILE.DAT"
$ FILENAME = F$EXTRACT(0,F$LOCATE(".",P1),P1)
```

> This portion of a command procedure shows how to locate a character within a string, and how to extract a substring ending at that location.
>
> The lexical function F$LOCATE gives the numeric value representing the offset position of a period in the character string value of P1. (The offset position of the period is equal to the length of the substring before the period.)
>
> This F$LOCATE function is used as an argument in the F$EXTRACT function to specify the number of characters to extract from the string. If a procedure is invoked with the parameter MYFILE.DAT, these statements result in the symbol FILENAME being given the value MYFILE.
>
> Note that the F$LOCATE function in the above example assumes that the file specification does not contain a node name or a directory specification containing a subdirectory name. To obtain the file name from a full file specification, use the F$PARSE function.

3
```
$ IF F$EXTRACT(12,2,F$TIME()) .GES. "12" THEN GOTO AFTERNOON
$ MORNING:
$ WRITE SYS$OUTPUT "Good morning!"
$ EXIT
$ AFTERNOON:
$ WRITE SYS$OUTPUT "Good afternoon!"
$ EXIT
```

> This example shows a procedure that displays a different message, depending on whether the current time is morning or afternoon. It first obtains the current time of day by using the F$TIME function. The F$TIME function returns a character string, which is the string argument for the F$EXTRACT function. The F$TIME function is automatically evaluated when it is used as an argument, so you do not need to use quotation marks.
>
> Next, the F$EXTRACT function extracts the hours from the date and time string returned by F$TIME. The string returned by F$TIME always contains the hours field beginning at an offset of 12 characters from the start of the string.
>
> The F$EXTRACT function extracts 2 characters from the string, beginning at this offset, and compares the string value extracted with the string value 12. If the comparison is true, then the procedure writes "Good afternoon!". Otherwise, it writes "Good morning!".
>
> Note that you can also use the F$CVTIME function to extract the hour field from a time specification. This method is easier than the one shown in the above example.

F$FAO

Converts character and numeric input to ASCII character strings. (FAO stands for formatted ASCII output.) By specifying formatting instructions, you can use the F$FAO function to convert integer values to character strings, to insert carriage returns and form feeds, to insert text, and so on.

FORMAT

F$FAO*(control-string[,argument[,...]])*

return value

A character string containing formatted ASCII output. This output string is created from the fixed text and FAO directives in the control string.

ARGUMENTS

control-string

Specifies the fixed text of the output string, consisting of text and any number of FAO directives. The control string may be any length. Specify the control string as a character string expression.

The F$FAO function uses FAO directives to modify or insert ASCII data into the fixed text in the control string.

Table DCL1–4 lists the FAO directives you can specify in a control string.

argument[,...]

Specifies from 1 to 15 arguments required by the FAO directives used in the control string. Specify the arguments as integer or character string expressions. Table DCL1–4 lists the argument types required by each FAO directive.

FAO directives may require one or more arguments. The order of the arguments must correspond exactly with the order of the directives in the control string. In most cases, an error message is not displayed if you misplace an argument.

If you specify an argument whose type (integer or string) does not match that of the corresponding directive, unpredictable results are returned. You can use the F$INTEGER and F$STRING lexical functions to convert arguments to the proper type.

If there are not enough arguments listed, F$FAO continues reading past the end of an argument list. Therefore, always be sure to include enough arguments to satisfy the requirements of all the directives in a control string.

If you specify an invalid parameter for any directive, you may see unexpected errors, which indicate that the command did not succeed. (These errors are passed through to you from the $FAO system service.)

DESCRIPTION The F$FAO lexical function invokes the $FAO system service to convert character and numeric input to ASCII character strings. (FAO stands for formatted ASCII output.) By specifying formatting instructions, you can use the F$FAO function to convert integer values to character strings, to insert carriage returns and form feeds, to insert text, and so on.

Specify an FAO directive using any one of the following formats:

Format	Function
!DD	One directive
!n(DD)	A directive repeated a specified number of times
!lengthDD	A directive that places its output in a field of a specified length
!n(lengthDD)	A directive that is repeated a specified number of times and generates output fields of a specified length

The exclamation point (!) indicates that the following character or characters are to be interpreted as an FAO directive. *DD* represents a 1- or 2-character uppercase code indicating the action that F$FAO is to perform. When specifying repeat counts, *n* is a decimal value specifying the number of times the directive is to be repeated. The *length* value is a decimal number that instructs F$FAO to generate an output field of "length" characters.

Repeat counts and output lengths may also be specified by using a number sign (#) in place of an absolute numeric value. If you use a number sign, you must specify the numeric value as an integer expression in the corresponding place in the argument list.

When a variable output field is specified with a repeat count, only one length parameter is required, because each output string has the specified length.

The FAO directives are grouped in the following categories:

- Character string insertion
- Zero-filled numeric conversion
- Blank-filled numeric conversion
- Special formatting
- Parameter interpretation

Table DCL1–4 summarizes the FAO directives and shows the required argument types. In addition, the following sections describe output strings from directives that perform character string insertion, zero-filled numeric conversion, and blank-filled numeric conversion.

Table DCL1–4   Summary of FAO Directives

Directive	Argument Type	Description
Character string insertion:		
!AS	String	Inserts a character string as is.
Zero-filled numeric conversion:		
!OB	Integer	Converts a byte to octal notation.
!OW	Integer	Converts a word to octal notation.
!OL	Integer	Converts a longword to octal notation.
!XB	Integer	Converts a byte to hexadecimal notation.
!XW	Integer	Converts a word to hexadecimal notation.
!XL	Integer	Converts a longword to hexadecimal notation.
!ZB	Integer	Converts a byte to decimal notation.
!ZW	Integer	Converts a word to decimal notation.
!ZL	Integer	Converts a longword to decimal notation.
Blank-filled numeric conversion:		
!UB	Integer	Converts a byte to decimal notation without adjusting for negative numbers.
!UW	Integer	Converts a word to decimal notation without adjusting for negative numbers.
!UL	Integer	Converts a longword to decimal notation without adjusting for negative numbers.
!SB	Integer	Converts a byte to decimal notation with negative numbers converted properly.
!SW	Integer	Converts a word to decimal notation with negative numbers converted properly.
!SL	Integer	Converts a longword to decimal notation with negative numbers converted properly.
Special formatting:		
!/	None	Inserts a carriage return and a line feed.
!_	None	Inserts a tab.
!^	None	Inserts a form feed.
!!	None	Inserts an exclamation point (!).
!%I	Integer	Converts a longword integer to a named UIC in the format [group-identifier,member-identifier].
!%S	None	Inserts an "s" if the most recently converted number is not 1. (Not recommended for use with multilingual products.)

Table DCL1–4 (Cont.)   Summary of FAO Directives

Directive	Argument Type	Description
!%U	Integer	Converts a longword integer to a numeric UIC in the format [g,m], where *g* is the group number and *m* is the member number.
		The directive inserts the brackets and the comma.
!n<...!>	None	Left-justifies and blank-fills all data represented by the instructions . . . in fields *n* characters wide.
!n*c	None	Repeats the character represented by *c* for *n* times.
!n%C	String	Inserts a character string when the most recently evaluated argument has the value *n*. (Recommended for use with multilingual products.)
!%E	String	Inserts a character string when the value of the most recently evaluated argument does not match any preceding !n%C directives. (Recommended for use with multilingual products.)
!%F	None	Marks the end of a plurals statement.
!%T	Integer equal to 0	Inserts the current time.
!%D	Integer equal to 0	Inserts the current date/time.
Argument interpretation:		
!-	None	Reuses the last argument.
!+	None	Skips the next argument.

Output Strings from Character String Insertion

The !AS directive inserts a character string (specified as an argument for the directive) into the control string. The field length of the character string when it is inserted into the control string defaults to the length of the character string. If the default length is shorter than an explicitly stated field length, the string is left-justified and blank-filled. If the default length is longer than an explicitly stated field length, the string is truncated on the right.

Output Strings from Zero-Filled Numeric Conversion

Directives for zero-filled numeric conversion convert an integer (specified as an argument for the directive) to decimal, octal, or hexadecimal notation. The ASCII representation of the integer is inserted into the control string. Default output field lengths for the converted argument are determined as follows.

Directives that convert arguments to octal notation return 3 digits for byte conversion, 6 digits for word conversion, and 11 digits for longword conversion. Numbers are right-justified and zero-filled on the left. Explicit-length fields longer than the default are blank-filled on the left. Explicit-length fields shorter than the default are truncated on the left.

Directives that convert arguments to hexadecimal notation return
2 digits for byte conversion, 4 digits for word conversion, and 8 digits
for longword conversion. Numbers are right-justified and zero-filled on the
left. Explicit-length fields longer than the default are blank-filled on the
left. Explicit-length fields shorter than the default are truncated on the
left.

Directives that convert arguments to decimal notation return the required
number of characters for the decimal number. Explicit-length fields longer
than the default are zero-filled on the left. If an explicit-length field is
shorter than the number of characters required for the decimal number,
the output field is completely filled with asterisks (*).

For byte conversion, only the low-order 8 bits of the binary representation
of the argument are used. For word conversion, only the low-order 16 bits
of the binary representation of the argument are used. For longword
conversion, the entire 32-bit binary representation of the argument is
used.

Output Strings from Blank-Filled Numeric Conversion

Directives for blank-filled numeric conversion convert an integer (specified
as an argument for the directive) to decimal notation. These directives
can convert the integer as a signed or unsigned number. The ASCII
representation of the integer is inserted into the control string.

Output field lengths for the converted argument default to the required
number of characters. Values shorter than explicit-length fields are right-
justified and blank-filled; values longer than explicit-length fields cause
the field to be filled with asterisks.

For byte conversion, only the low-order 8 bits of the binary representation
of the argument are used. For word conversion, only the low-order 16 bits
of the binary representation of the argument are used. For longword
conversion, the entire 32-bit binary representation of the argument is
used.

Output Strings from Special Formatting Directives

The !n%C and !%E directives insert an ASCII string (based on the value
of the most recently evaluated argument) into the output string. These
directives are useful for inserting irregular plural nouns and verbs.

If the most recently evaluated argument equals n, the text between one
directive and the next is inserted into the output string. If the most
recently evaluated argument does not equal n, the next !n%C directive is
processed.

If n must be a negative number, you must specify it as an argument and
use the number sign (#).

You can specify the !n%C and !%E directives with repeat counts. If you
specify repeat counts, the text between one directive and the next is copied
to the output string the specified number of times.

The %F directive marks the end of a plurals statement.

EXAMPLES

1
```
$ COUNT = 57
$ REPORT = F$FAO("NUMBER OF FORMS = !SL",COUNT)
$ SHOW SYMBOL REPORT
$ REPORT = "NUMBER OF FORMS = 57"
```

In this command procedure, the FAO directive !SL is used in a control string to convert the number equated to the symbol COUNT to a character string. The converted string is inserted into the control string.

Note that COUNT is assigned an integer value of 57. The F$FAO function returns the ASCII string, "NUMBER OF FORMS = 57", and assigns the string to the symbol REPORT.

2
```
$ A = "ERR"
$ B = "IS"
$ C = "HUM"
$ D = "AN"
$ PHRASE = F$FAO("TO !3(AS)",A,B,C+D)
$ SHOW SYMBOL PHRASE
$ PHRASE = "TO ERRISHUMAN"
```

In this command procedure, the !AS directive is used to insert the values assigned to the symbols A, B, C, and D into the control string.

Because the specified repeat count for the !AS directive is 3, F$FAO looks for three arguments. The arguments in this example include the symbol A ("ERR"), the symbol B ("IS"), and the expression C+D ("HUMAN"). Note that the values of these string arguments are concatenated to form the string "ERRISHUMAN".

3
```
$ A = "ERR"
$ B = "IS"
$ C = "HUMAN"
$ PHRASE = F$FAO("TO !#(#AS)",3,6,A,B,C)
$ SHOW SYMBOL PHRASE
$ PHRASE = "TO ERR   IS    HUMAN "
```

In this command procedure, the F$FAO function is used with the !AS directive to format a character string. The first number sign (#) represents the repeat count given by the first argument, 3. The second number sign represents the field size given by the second argument, 6. The next three arguments (A,B,C) provide the strings that are placed into the control string each time the !AS directive is repeated.

Each argument string is output to a field having a length of 6 characters. Because each string is less than 6 characters, each field is left-justified and padded with blank spaces. The resulting string is assigned to the symbol PHRASE.

4   $ OFFSPRING = 1
    $ REPORT = F$FAO -
    -$ ("There !0UL!1%Cis!Eare!%F !-!UL !1%Cchild!%Echildren!%Fhere",'OFFSPRING')
    $ SHOW SYMBOL REPORT
    $ REPORT ="There is 1 child here"

> In this command procedure, the !0UL directive evaluates the argument
> OFFSPRING but does not insert the value in the output string. The !n%C
> directive inserts the character string "is" into the output string because its
> value and the value of the argument OFFSPRING match. The directives
> !-!UL evaluate the argument a second time so that the correct character
> string can be inserted in the proper place in the output string. The !%F
> directive marks the end of each plurals statement. The F$FAO function
> returns the ASCII string "There is 1 child here" and assigns the string to
> the symbol REPORT.

F$FILE_ATTRIBUTES

Returns attribute information for a specified file.

FORMAT	**F$FILE_ATTRIBUTES**(filespec,item)

return value   Either an integer or a character string, depending on the item you request. Table DCL1-5 shows the data types of the values returned for each item.

ARGUMENTS   **filespec**
Specifies the name of the file about which you are requesting information. You must specify the file name as a character string expression.

You can specify only one file name. Wildcard characters are not allowed.

item
Indicates which attribute of the file is to be returned. The **item** argument must be specified as a character string expression, and can be any one of the VMS RMS field names listed in Table DCL1-5.

DESCRIPTION   Use the F$FILE_ATTRIBUTES lexical function in DCL assignment statements and expressions to return file attribute information. Table DCL1-5 lists the items you can specify with the F$FILE_ATTRIBUTES function, the information returned, and the data type of this information.

Table DCL1-5   F$FILE_ATTRIBUTES Items

Item	Return Type	Information Returned
AI	String	TRUE if after-image (AI) journaling is enabled; returns TRUE or FALSE
ALQ	Integer	Allocation quantity
BDT	String	Backup date/time
BI	String	TRUE if before-image (BI) journaling is enabled; returns TRUE or FALSE
BKS	Integer	Bucket size
BLS	Integer	Block size
CBT	String	TRUE if contiguous-best-try; returns TRUE or FALSE
CDT	String	Creation date/time
CTG	String	TRUE if contiguous; returns TRUE or FALSE

Table DCL1–5 (Cont.)   F$FILE_ATTRIBUTES Items

Item	Return Type	Information Returned
DEQ	Integer	Default extension quantity
DID	String	Directory ID string
DVI	String	Device name string
EDT	String	Expiration date/time
EOF	Integer	Number of blocks used
ERASE	String	TRUE if a file's contents are erased before a file is deleted; returns TRUE or FALSE
FFB	Integer	First free byte
FID	String	File ID string
FSZ	Integer	Fixed control area size
GBC	Integer	Global buffer count
GRP	Integer	Owner group number
JOURNAL_FILE	String	TRUE if the file is a journal; returns TRUE or FALSE
KNOWN	String	Known file; returns TRUE or FALSE to indicate whether file is installed with the Install Utility (INSTALL)
LOCKED	String	TRUE if a file is deaccessed-locked; returns TRUE or FALSE
MBM	Integer	Owner member number
MRN	Integer	Maximum record number
MRS	Integer	Maximum record size
NOA	Integer	Number of areas
NOBACKUP	String	FALSE if the file is marked for backup; TRUE if the file is marked NOBACKUP
NOK	Integer	Number of keys
ORG	String	File organization; returns SEQ, REL, IDX
PRO	String	File protection string
PVN	Integer	Prolog version number
RAT	String	Record attributes; returns CR, PRN, FTN, " "
RCK	String	TRUE if read check; returns TRUE, FALSE
RDT	String	Revision date/time
RFM	String	Record format string; returns the values VAR, FIX, VFC, UDF, STM, STMLF, STMCR
RU	String	TRUE if recovery unit (RU) journaling is enabled; returns TRUE or FALSE

Table DCL1-5 (Cont.)   F$FILE_ATTRIBUTES Items

Item	Return Type	Information Returned
RVN	Integer	Revision number
STORED_SEMANTICS	String	ASCII string that represents stored semantics
UIC	String	Owner user identification code (UIC) string
WCK	String	TRUE if write check; returns TRUE, FALSE

File attributes are stored in the file header, which is created from information in VMS RMS control blocks. For more information on VMS RMS control blocks, see the *VMS Record Management Services Manual*.

EXAMPLES

1
```
$  FILE_ORG = F$FILE_ATTRIBUTES("QUEST.DAT","ORG")
$  SHOW SYMBOL FILE_ORG
   FILE_ORG = "SEQ"
```

> This example uses the F$FILE_ATTRIBUTES function to assign the value of the file organization type to the symbol FILE_ORG. The F$FILE_ATTRIBUTES function returns the character string SEQ to show that QUEST.DAT is a sequential file.
>
> The QUEST.DAT and ORG arguments for the F$FILE_ATTRIBUTES function are string literals and must be enclosed in quotation marks (" ") when used in expressions.

2
```
$  RFM = F$FILE_ATTRIBUTES("KANSAS::USE$:[CARS]SALES.CMD","RFM")
$  SHOW SYMBOL RFM
   RFM = "VAR"
```

> This example uses the F$FILE_ATTRIBUTES function to return information about a file on a remote node. The function returns the record format string VAR, indicating that records are variable length.

F$GETDVI

Returns a specified item of information for a specified device.

FORMAT	**F$GETDVI**(device-name,item)

return value
Either an integer or a character string, depending on the item you request. Table DCL1–6 shows the data types of the values returned for each item.

ARGUMENTS

device-name
Specifies a physical device name or a logical name equated to a physical device name. Specify the device name as a character string expression.

After the **device-name** argument is evaluated, the F$GETDVI function examines the first character of the name. If the first character is an underscore (_), the name is considered a physical device name. Otherwise, a single level of logical name translation is performed and the equivalence name, if any, is used.

item
Specifies the type of device information to be returned. The **item** argument must be specified as a character string expression and can be any one of the items listed in Table DCL1–6.

DESCRIPTION
The F$GETDVI lexical function invokes the $GETDVI system service to return a specified item of information for a specified device. You can obtain a list of devices on your current system by using the lexical function F$DEVICE.

This lexical function allows a process to obtain information for a device to which the process has not necessarily assigned a channel.

The F$GETDVI function returns information on all items that can be specified with the $GETDVI system service. In addition to the items that the $GETDVI system service allows, the F$GETDVI function allows you to specify the item EXISTS.

Table DCL1–6 lists the items you can specify with the F$GETDVI function, the type of information returned, and the data types of the return values. In addition to the return information listed in Table DCL1–6, the F$GETDVI lexical function returns any error messages generated by the system service $GETDVI. Table DCL1–7 lists the values returned by the DEVCLASS item. Table DCL1–8 lists the values returned by the DEVTYPE item. For more information on the $GETDVI system service and the items you can specify, see the *VMS System Services Reference Manual*.

Table DCL1–6   F$GETDVI Items

Item	Return Type	Information Returned†
ACPPID	String	Ancillary control process (ACP) identification.
ACPTYPE	String	ACP type code, as one of the following strings: F11V1, F11V2, JNL, MTA, NET, REM, or ILLEGAL if the device is not mounted or is mounted using the /FOREIGN qualifier.
ALL	String	TRUE or FALSE to indicate whether the device is allocated.
ALLDEVNAM	String	Allocation class device name.
ALLOCLASS	Longword integer between 0 and 255	Allocation class of the host.
ALT_HOST_AVAIL	String	TRUE or FALSE to indicate whether the host serving the alternate path is available.
ALT_HOST_NAME	String	Name of the host serving the alternate path.
ALT_HOST_TYPE	String	Hardware type of the host serving the alternate path.
AVL	String	TRUE or FALSE to indicate whether the device is available for use.
CCL	String	TRUE or FALSE to indicate whether the device is a carriage control device.
CLUSTER	Integer	Volume cluster size.
CONCEALED	String	TRUE or FALSE to indicate whether the logical device name translates to a concealed device.
CYLINDERS	Integer	Number of cylinders on the volume (disks only).
DEVBUFSIZ	Integer	Device buffer size.
DEVCHAR	Integer	Device characteristics.
DEVCHAR2	Integer	Additional device characteristics.
DEVCLASS	Integer	Device class. (See Table DCL1–7 for a list of the values returned.)
DEVDEPEND	Integer	Device-dependent information.
DEVDEPEND2	Integer	Additional device-dependent information.
DEVLOCKNAM	String	A unique lock name for the device.
DEVNAM	String	Device name.
DEVSTS	Integer	Device-dependent status information.
DEVTYPE	Integer	Device type. (See Table DCL1–8 for a list of the values returned.)
DFS_ACCESS	String	TRUE or FALSE to indicate whether the device is a virtual disk connected to a remote Distributed File System (DFS) server.
DIR	String	TRUE or FALSE to indicate whether the device is directory structured.
DMT	String	TRUE or FALSE to indicate whether the device is marked for dismount.

†In addition to the return information listed, the F$GETDVI lexical function returns any error messages generated by the system service $GETDVI.

Table DCL1-6 (Cont.)   F$GETDVI Items

Item	Return Type	Information Returned†
DUA	String	TRUE or FALSE to indicate whether the device is a generic device.
ELG	String	TRUE or FALSE to indicate whether the device has error logging enabled.
ERRCNT	Integer	Error count.
EXISTS	String	TRUE or FALSE to indicate whether the device exists on the system.
FOD	String	TRUE or FALSE to indicate whether the device is a files-oriented device.
FOR	String	TRUE or FALSE to indicate whether the device is mounted using the /FOREIGN qualifier.
FREEBLOCKS	Integer	Number of free blocks on the volume (disks only).
FULLDEVNAM	String	Fully qualified device name.
GEN	String	TRUE or FALSE to indicate whether the device is a generic device.
HOST_AVAIL	String	TRUE or FALSE to indicate whether the host serving the primary path is available.
HOST_COUNT	Integer	Number of hosts that make the device available to other nodes in the VAXcluster.
HOST_NAME	String	Name of the host serving the primary path.
HOST_TYPE	String	Hardware type of the host serving the primary path.
IDV	String	TRUE or FALSE to indicate whether the device is capable of providing input.
LOCKID	Integer	Clusterwide lock identification.
LOGVOLNAM	String	Logical volume name.
MAXBLOCK	Integer	Number of logical blocks on the volume.
MAXFILES	Integer	Maximum number of files on the volume (disks only).
MBX	String	TRUE or FALSE to indicate whether the device is a mailbox.
MEDIA_ID	String	Nondecoded media ID.
MEDIA_NAME	String	Either the name of the disk or the tape type.
MEDIA_TYPE	String	Device name prefix.
MNT	String	TRUE or FALSE to indicate whether the device is mounted.
MOUNTCNT	Integer	Mount count.
NET	String	TRUE or FALSE to indicate whether the device is a network device.
NEXTDEVNAM	String	Device name of the next volume in a volume set (disks only).
ODV	String	TRUE or FALSE to indicate whether the device is capable of providing output.
OPCNT	Integer	Operation count.
OPR	String	TRUE or FALSE to indicate whether the device is an operator.

†In addition to the return information listed, the F$GETDVI lexical function returns any error messages generated by the system service $GETDVI.

(continued on next page)

Table DCL1–6 (Cont.)   F$GETDVI Items

Item	Return Type	Information Returned†
OWNUIC	String	User identification code (UIC) of the device owner.
PID	String	Process identification number of the device owner.
RCK	String	TRUE or FALSE to indicate whether the device has read checking enabled.
RCT	String	TRUE or FALSE to indicate whether the disk contains RCT.
REC	String	TRUE or FALSE to indicate whether the device is record oriented.
RECSIZ	Integer	Blocked record size.
REFCNT	Integer	Reference count of processes using the device.
REMOTE_DEVICE	String	TRUE or FALSE to indicate whether the device is a remote device.
RND	String	TRUE or FALSE to indicate whether the device allows random access.
ROOTDEVNAM	String	Device name of the root volume in a volume set (disks only).
RTM	String	TRUE or FALSE to indicate whether the device is a real-time device.
SDI	String	TRUE or FALSE to indicate whether the device is single-directory structured.
SECTORS	Integer	Number of sectors per track (disks only).
SERIALNUM	Integer	Volume serial number (disks only).
SERVED_DEVICE	String	TRUE or FALSE to indicate whether the device is a served device.
SET_HOST_TERMINAL	String	TRUE or FALSE to indicate whether the device is a remote terminal for a SET HOST session from a remote node.
SHDW_CATCHUP_ COPYING	String	TRUE or FALSE to indicate whether the device is a member that is the target of a full copy operation.
SHDW_MASTER	String	TRUE or FALSE to indicate whether the device is a virtual unit.
SHDW_MASTER_NAME	String	Device name of the virtual unit that represents the shadow set of which the specified device is a member. F$GETDVI returns a null string ("") if the specified device is not a member, or is itself a virtual unit.
SHDW_MEMBER	String	TRUE or FALSE to indicate whether the device is a shadow set member.
SHDW_MERGE_COPYING	String	TRUE or FALSE to indicate whether the device is a merge member of the shadow set.

†In addition to the return information listed, the F$GETDVI lexical function returns any error messages generated by the system service $GETDVI.

Table DCL1–6 (Cont.)   F$GETDVI Items

Item	Return Type	Information Returned†
SHDW_NEXT_MBR_NAME	String	Device name of the next member in the shadow set. If you specify a virtual unit or a member, F$GETDVI returns the device name of members in random order. If you specify the name of a device that is neither a virtual unit nor a member, F$GETDVI returns a null string.
		F$GETDVI returns the device name of the next member in the shadow set even if the next member has been removed from the shadow set.
		The device name includes the allocation class if the allocation class is not zero; otherwise it includes the device name of the disk controller.
SHR	String	TRUE or FALSE to indicate whether the device is shareable.
SPL	String	TRUE or FALSE to indicate whether the device is being spooled.
SPLDEVNAM	String	Name of the device being spooled.
SQD	String	TRUE or FALSE to indicate whether the device is sequential block-oriented (that is, magnetic tape).
STS	Integer	Status information.
SWL	String	TRUE or FALSE to indicate whether the device is software write-locked.
TRACKS	Integer	Number of tracks per cylinder (disks only).
TRANSCNT	Integer	Volume transaction count.
TRM	String	TRUE or FALSE to indicate whether the device is a terminal.
TT_ACCPORNAM	String	The terminal server name and port name.
TT_ALTYPEAHD	String	TRUE or FALSE to indicate whether the terminal has an alternate type-ahead buffer (terminals only).
TT_ANSICRT	String	TRUE or FALSE to indicate whether the terminal is an ANSI CRT terminal (terminals only).
TT_APP_KEYPAD	String	TRUE or FALSE to indicate whether the keypad is in applications mode (terminals only).
TT_AUTOBAUD	String	TRUE or FALSE to indicate whether the terminal has automatic baud rate detection (terminals only).
TT_AVO	String	TRUE or FALSE to indicate whether the terminal has a VT100-family terminal display (terminals only).
TT_BLOCK	String	TRUE or FALSE to indicate whether the terminal has block mode capability (terminals only).
TT_BRDCSTMBX	String	TRUE or FALSE to indicate whether the terminal uses mailbox broadcast messages (terminals only).
TT_CRFILL	String	TRUE or FALSE to indicate whether the terminal requires fill after a carriage return (terminals only).

†In addition to the return information listed, the F$GETDVI lexical function returns any error messages generated by the system service $GETDVI.

Table DCL1–6 (Cont.)   F$GETDVI Items

Item	Return Type	Information Returned†
TT_DECCRT	String	TRUE or FALSE to indicate whether the terminal is a Digital CRT terminal (terminals only).
TT_DECCRT2	String	TRUE or FALSE to indicate whether the terminal is a Digital CRT2 terminal (terminals only).
TT_DECCRT3	String	TRUE or FALSE to indicate whether the terminal is a Digital CRT3 terminal (terminals only).
TT_DECCRT4	String	TRUE or FALSE to indicate whether the terminal is a Digital CRT4 terminal (terminals only).
TT_DIALUP	String	TRUE or FALSE to indicate whether the terminal is connected to dialup (terminals only).
TT_DISCONNECT	String	TRUE or FALSE to indicate whether the terminal can be disconnected (terminals only).
TT_DMA	String	TRUE or FALSE to indicate whether the terminal has direct memory access (DMA) mode (terminals only).
TT_DRCS	String	TRUE or FALSE to indicate whether the terminal supports loadable character fonts (terminals only).
TT_EDIT	String	TRUE or FALSE to indicate whether the edit characteristic is set.
TT_EDITING	String	TRUE or FALSE to indicate whether advanced editing is enabled (terminals only).
TT_EIGHTBIT	String	TRUE or FALSE to indicate whether the terminal uses the 8-bit ASCII character set (terminals only).
TT_ESCAPE	String	TRUE or FALSE to indicate whether the terminal generates escape sequences (terminals only).
TT_FALLBACK	String	TRUE or FALSE to indicate whether the terminal uses the multinational fallback option (terminals only).
TT_HALFDUP	String	TRUE or FALSE to indicate whether the terminal is in half-duplex mode (terminals only).
TT_HANGUP	String	TRUE or FALSE to indicate whether the hangup characteristic is set (terminals only).
TT_HOSTSYNC	String	TRUE or FALSE to indicate whether the terminal has host/terminal communication (terminals only).
TT_INSERT	String	TRUE or FALSE to indicate whether insert mode is the default line editing mode (terminals only).
TT_LFFILL	String	TRUE or FALSE to indicate whether the terminal requires fill after a line feed (terminals only).
TT_LOCALECHO	String	TRUE or FALSE to indicate whether the local echo characteristic is set (terminals only).
TT_LOWER	String	TRUE or FALSE to indicate whether the terminal has the lowercase characters set (terminals only).

†In addition to the return information listed, the F$GETDVI lexical function returns any error messages generated by the system service $GETDVI.

Table DCL1–6 (Cont.)   F$GETDVI Items

Item	Return Type	Information Returned†
TT_MBXDSABL	String	TRUE or FALSE to indicate whether mailboxes associated with the terminal will receive unsolicited input notification or input notification (terminals only).
TT_MECHFORM	String	TRUE or FALSE to indicate whether the terminal has mechanical form feed (terminals only).
TT_MECHTAB	String	TRUE or FALSE to indicate whether the terminal has mechanical tabs and is capable of tab expansion (terminals only).
TT_MODEM	String	TRUE or FALSE to indicate whether the terminal is connected to a modem (terminals only).
TT_MODHANGUP	String	TRUE or FALSE to indicate whether the modify hangup characteristic is set (terminals only).
TT_NOBRDCST	String	TRUE or FALSE to indicate whether the terminal will receive broadcast messages (terminals only).
TT_NOECHO	String	TRUE or FALSE to indicate whether the input characters are echoed.
TT_NOTYPEAHD	String	TRUE or FALSE to indicate whether data must be solicited by a read operation.
TT_OPER	String	TRUE or FALSE to indicate whether the terminal is an operator terminal (terminals only).
TT_PAGE	Integer	Terminal page length (terminals only).
TT_PASTHRU	String	TRUE or FALSE to indicate whether PASSALL mode with flow control is available (terminals only).
TT_PHYDEVNAM	String	Physical device name associated with a channel number or virtual terminal.
TT_PRINTER	String	TRUE or FALSE to indicate whether there is a printer port available (terminals only).
TT_READSYNC	String	TRUE or FALSE to indicate whether the terminal has read synchronization (terminals only).
TT_REGIS	String	TRUE or FALSE to indicate whether the terminal has ReGIS graphics (terminals only).
TT_REMOTE	String	TRUE or FALSE to indicate whether the terminal has established modem control (terminals only).
TT_SCOPE	String	TRUE or FALSE to indicate whether the terminal is a video screen display (terminals only).
TT_SECURE	String	TRUE or FALSE to indicate whether the terminal can recognize the secure server (terminals only).
TT_SETSPEED	String	TRUE or FALSE to indicate whether you cannot set the speed on the terminal line (terminals only).
TT_SIXEL	String	TRUE or FALSE to indicate whether the sixel is supported (terminals only).

†In addition to the return information listed, the F$GETDVI lexical function returns any error messages generated by the system service $GETDVI.

Lexical Functions
F$GETDVI

Table DCL1–6 (Cont.)   F$GETDVI Items

Item	Return Type	Information Returned†
TT_SYSPWD	String	TRUE or FALSE to indicate whether the system password is enabled for a particular terminal.
TT_TTSYNC	String	TRUE or FALSE to indicate whether there is terminal/host synchronization (terminals only).
TT_WRAP	String	TRUE or FALSE to indicate whether a new line should be inserted if the cursor moves beyond the right margin.
UNIT	Integer	The unit number.
VOLCOUNT	Integer	The count of volumes in a volume set (disks only).
VOLNAM	String	The volume name.
VOLNUMBER	Integer	Number of the current volume in a volume set (disks only).
VOLSETMEM	String	TRUE or FALSE to indicate whether the device is a volume set (disks only).
VPRO	String	The volume protection mask.
WCK	String	TRUE or FALSE to indicate whether the device has write checking enabled.

†In addition to the return information listed, the F$GETDVI lexical function returns any error messages generated by the system service $GETDVI.

Table DCL1–7 lists the values returned by the DEVCLASS item.

Table DCL1–7   Values Returned by the DEVCLASS Item

Value	Device Class	Symbolic Name
1	Disk device	DC$_DISK
2	Tape device	DC$_TAPE
32	Synchronous communications device	DC$_SCOM
65	Card reader	DC$_CARD
66	Terminal	DC$_TERM
67	Line printer	DC$_LP
70	Workstation	DC$_WORKSTATION
96	Real-time	DC$_REALTIME
97	DECvoice	DC$_DECVOICE
128	Bus	DC$_BUS
160	Mailbox	DC$_ MAILBOX
161	Journal	DC$_JOURNAL
170	Remote console storage	DC$_REMCSL_STORAGE
200	Miscellaneous device	DC$_MISC

Table DCL1–8 lists the values returned by the DEVTYPE item.

Table DCL1–8 Values Returned by the DEVTYPE Item

Value	Device Type	Value	Device Type
Device Class: DC$_DISK			
1	RK06	32	RD54
2	RK07	33	CRX50
3	RP04	33	RX50
4	RP05	34	RRD50
5	RP06	35	GENERIC_DU
6	RMO3	36	RX33
7	RP07	37	RX18
8	RP07HT	38	RA70
9	RL01	39	RA90
10	RL02	40	RD32
11	RX02	41	DISK9
12	RX04	42	RX35
13	RM80	43	RF30
14	TU58	44	RF70
15	RM05	45	RD33
16	RX01	46	ESE20
17	ML11	47	TU56
18	RB02	48	RZ22
19	RB80	49	RZ23
20	RA80	50	RZ24
21	RA81	51	RZ55
22	RA60	52	RRD40
23	RC25	54	GENERIC_DK
23	RZ01	55	RX23
24	RZF01	129	FD1
25	RD51	130	FD2
26	RX50	131	FD3
27	RD52	132	FD4
28	RD53	133	FD5
29	RD26	134	FD6
30	RA82	135	FD7
31	RD31	136	FD8

Table DCL1–8 (Cont.)   Values Returned by the DEVTYPE Item

Value	Device Type	Value	Device Type
Device Class: DC$_TAPE			
1	TE16	14	MW_TSV05
2	TU45	15	TK70
3	TU77	16	RV20
4	TS11	16	RV80
5	TU78	17	TK60
6	TA78	18	GENERIC_TU
7	TU80	19	TA79
8	TU81	20	TAPE9
9	TA81	21	TA90
10	TK50	22	TF30
11	MR_TU70	23	TF70
12	MR_TU72	24	RV60
13	MW_TSU05		
Device Class: DC$_SCOM			
1	DMC11	18	YQ_3271
2	DMR11	19	YR_DDCMP
3	XK_3271	20	YS_SDLC
4	XJ_2780	21	UK_KTC32
5	NW_X25	22	DEQNA
6	NV_X29	23	DMV11
7	SB_ISB11	24	ES_LANCE
8	MX_MUX200	25	DELUA
9	DMP11	26	NQ_3271
10	DMF32	27	DMB32
11	XV_3271	28	YI_KMS11K
12	CI	29	ET_DEBNT
13	NI	29	ET_DEBNA
14	UNA11	30	SJ_DSV11
14	DEUNA	31	SL_DSB32
15	YN_X25	32	ZS_DST32
16	YO_X25	33	XQ_DELQA
17	YP_ADCCP		

Table DCL1–8 (Cont.)   Values Returned by the DEVTYPE Item

Value	Device Type	Value	Device Type
Device Class: DC$_CARD			
1	CR11		
Device Class: DC$_TERM			
0	TTYUNKN	37	LA24
1	VT05	37	LA100
2	VK100	38	LQP02
3	VT173	40	LA210
64	VT5X	41	LN03
10	TEK401X	42	LN01K
16	FT1	43	LA80
17	FT2	64	VT52
18	FT3	65	VT55
19	FT4	96	VT100
20	FT5	97	VT101
21	FT6	98	VT102
22	FT7	99	VT105
23	FT8	100	VT125
32	LAX	101	VT131
32	LA36	102	VT132
33	LA120	110	VT200_Series
34	LA34	111	Pro_Series
35	LA38	112	VT300_Series
36	LA12	113	VT400_Series
Device Class: DC$_LP			
1	LP11	4	LC_DMF32
2	LA11	5	LI_DMB32
3	LA180	6	PRTR9
Device Class: DC$_WORKSTATION			
1	VAXstation 100	4	VAXstation virtual device
2	VAXstation 125	5	DECW output device
3	VAXstation 8200	6	DECW input device

Table DCL1–8 (Cont.)   Values Returned by the DEVTYPE Item

Value	Device Type	Value	Device Type
Device Class: DC$_REALTIME			
1	LPA11	9	XP_PCL11B
2	DR780	10	IX_IEX11
3	DR750	11	FP_FEPCM
4	DR11W	12	TK_FCM
5	PCL11R	13	XI_DR11C
6	PCL11T	14	XA_DRV11WA
7	DR11C	15	DRB32
8	XI_DR11C	16	HX_DRQ3B
Device Class: DC$_DECVOICE			
1	VQ class driver	3	VM M3135 port driver
2	VX M7132 port driver	4	VN M3136 port driver
Device Class: DC$_BUS			
1	CI780	15	BCI750
2	CI750	16	BCA
3	UQPORT	17	RQDX3
3	UDA50	18	NISCA
4	UDA50A	19	AIO
5	LESI	20	AIE
6	TU81P	20	DEBNT
7	RDRX	21	BSA
8	TK50P	21	KSB50
9	RUX50P	22	TK70P
10	RC26P	23	RV20P
11	QDA50	23	RV80P
11	KDA50	24	TK60P
12	BDA50	25	SII
12	KDB50	26	KFSQSA
13	RRD50P	27	SHAC
14	QDA25	28	CIXCA
14	KDA25	29	CIXCB

Table DCL1–8 (Cont.)   Values Returned by the DEVTYPE Item

Value	Device Type	Value	Device Type
Device Class: DC$_MAILBOX			
1	MBX		
2	SHRMBX		
3	NULL		
Device Class: DC$_JOURNAL			
0	UNKNJNL	3	AIJNL
1	RUJNL	4	ATJNL
2	BIJNL	5	CLJNL
Device Class: DC$_REMCSL_STORAGE			
DAP-accessed device	1		
Device Class: DC$_MISC			
1	DN11	3	SFUN9
2	PV	4	USER9

EXAMPLE

```
$  ERR = F$GETDVI("_DQA0","ERRCNT")
$  SHOW SYMBOL ERR
  ERR = 0  Hex = 00000000 Octal = 000000
```

This example shows how to use the F$GETDVI function to return an error count for the device DQA0. You must place quotation marks (" ") around the device name DQA0 and the item ERRCNT because they are string literals.

F$GETJPI

Returns accounting, status, and identification information for the specified process.

Requires GROUP privilege to obtain information on other processes in the same group. Requires WORLD privilege to obtain information on any other processes in the system.

FORMAT

F$GETJPI(*pid,item*)

return value

Either an integer or a character string, depending on the item you request. Table DCL1-9 shows the data types of the values returned for each item.

ARGUMENTS

pid

Specifies the process identification (PID) number of the process for which information is being reported. Specify the **pid** argument as a character string expression. You can omit the leading zeros.

If you specify a null string (""), the current PID number is used.

You cannot use a wildcard to specify the **pid** argument in the F$GETJPI function, as you can with the $GETJPI system service. To get a list of process identification numbers, use the F$PID function.

item

Indicates the type of process information to be returned. Specify the **item** argument as a character string expression. You can specify any one of the items listed in Table DCL1-9.

DESCRIPTION

The F$GETJPI lexical function invokes the $GETJPI system service to return accounting, status, and identification information for the specified process. The function returns information on all items that can be specified with the $GETJPI system service. For more information on the $GETJPI system service, see the *VMS System Services Reference Manual*.

Table DCL1-9 lists the items you can specify with the F$GETJPI function, the information returned, and the data type of this information.

Table DCL1–9   F$GETJPI Items

Item	Return Type	Information Returned
ACCOUNT	String	Account name string (8 characters filled with trailing blanks).
APTCNT	Integer	Active page table count.
ASTACT	Integer	Access modes with active asynchronous system traps (ASTs).
ASTCNT	Integer	Remaining AST quota.
ASTEN	Integer	Access modes with ASTs enabled.
ASTLM	Integer	AST limit quota.
AUTHPRI	Integer	Maximum priority that a process without the ALTPRI (alter priority) privilege can achieve with the $SETPRI system service.
AUTHPRIV	String	Privileges that a process is authorized to enable.
BIOCNT	Integer	Remaining buffered I/O quota.
BIOLM	Integer	Buffered I/O limit quota.
BUFIO	Integer	Count of process buffered I/O operations.
BYTCNT	Integer	Remaining buffered I/O byte count quota.
BYTLM	Integer	Buffered I/O byte count limit quota.
CLINAME	String	Current command language interpreter; always returns DCL.
CPULIM	Integer	Limit on process CPU time.
CPUTIM	Integer	CPU time used in hundredths of a second.
CREPRC_FLAGS	Integer	Flags specified by the **stsflg** argument in the $CREPRC call that created the process.
CURPRIV	String	Current process privileges.
DFPFC	Integer	Default page fault cluster size.
DFWSCNT	Integer	Default working set size.
DIOCNT	Integer	Remaining direct I/O quota.
DIOLM	Integer	Direct I/O limit quota.
DIRIO	Integer	Count of direct I/O operations for the process.
EFCS	Integer	Local event flags 0–31.
EFCU	Integer	Local event flags 32–63.
EFWM	Integer	Event flag wait mask.
ENQCNT	Integer	Lock request quota remaining.
ENQLM	Integer	Lock request quota limit.
EXCVEC	Integer	Address of a list of exception vectors.

Table DCL1–9 (Cont.) F$GETJPI Items

Item	Return Type	Information Returned
FAST_VP_SWITCH	Integer	Number of times this process has issued a vector instruction that enabled an inactive vector processor without the expense of a vector context switch.
FILCNT	Integer	Remaining open file quota.
FILLM	Integer	Open file quota.
FINALEXC	Integer	Address of a list of final exception vectors.
FREP0VA	Integer	First free page at end of program region (P0 space) (irrelevant if no image is running).
FREP1VA	Integer	First free page at end of control region (P1 space).
FREPTECNT	Integer	Number of pages available for virtual memory expansion.
GPGCNT	Integer	Global page count in working set.
GRP	Integer	Group number of the user identification code (UIC).
IMAGECOUNT	Integer	Number of images that have been run down for the process.
IMAGNAME	String	File name of the current image.
IMAGPRIV	String	Privileges with which the current image was installed.
JOBPRCCNT	Integer	Number of subprocesses owned by the job.
JOBTYPE	Integer	Execution mode of the process at the root of the job tree.
LAST_LOGIN_I	String	Time of your last interactive login (the value that was reported when you logged in).
LAST_LOGIN_N	String	Time of your last noninteractive login (the value that was reported when you logged in).
LOGIN_FAILURES	Integer	Number of login failures that occurred prior to the start of the current session (the value that was reported when you logged in).
LOGIN_FLAGS	Integer	A longword bitmask that contains additional information relating to the login sequence. (For more information, see the description of the $GETJPI system service in the VMS System Services Reference Manual.)
LOGINTIM	String	Process creation time.
MASTER_PID	String	Process identification (PID) number of the process at the top of the current job's process tree.

Table DCL1–9 (Cont.)   F$GETJPI Items

Item	Return Type	Information Returned
MAXDETACH	Integer	Maximum number of detached processes allowed the user who owns the process.
MAXJOBS	Integer	Maximum number of active processes allowed for the user who owns the process.
MEM	Integer	Member number of the UIC.
MODE	String	Current process mode (BATCH, INTERACTIVE, NETWORK, or OTHER).
MSGMASK	Integer	Default message mask.
OWNER	String	Process identification number of process owner.
PAGEFLTS	Integer	Count of page faults.
PAGFILCNT	Integer	Remaining paging file quota.
PAGFILLOC	Integer	Location of the paging file.
PGFLQUOTA	Integer	Paging file quota (maximum virtual page count).
PHDFLAGS	Integer	Flags word.
PID	String	Process identification number.
PPGCNT	Integer	Process page count.
PRCCNT	Integer	Number of subprocesses owned by the process.
PRCLM	Integer	Subprocess quota.
PRCNAM	String	Process name.
PRI	Integer	Process's current priority.
PRIB	Integer	Process's base priority.
PROC_INDEX	Integer	Process's index number.
PROCESS_RIGHTS	String	The contents of the process's local rights list, including your UIC. This item code returns a list of identifier names separated by commas (,).
PROCPRIV	String	Process's default privileges.
RIGHTSLIST	String	The contents of all of the process rights lists; the equivalent of PROCESS_RIGHTS plus SYSTEM_RIGHTS. This item code returns a list of identifier names separated by commas.
SHRFILLM	Integer	Maximum number of open shared files allowed for the job to which the process belongs.
SITESPEC	Integer	Per-process site-specific longword.

Table DCL1–9 (Cont.)   F$GETJPI Items

Item	Return Type	Information Returned
SLOW_VP_SWITCH	Integer	Number of times this process has issued a vector instruction that enabled an inactive vector processor with a full vector context switch.
STATE	String	Process state.
STS	Integer	Process status flags.
SWPFILLOC	Integer	Location of the swap file.
SYSTEM_RIGHTS	String	The contents of the system rights list for the process. This item code returns a list of identifier names separated by commas.
TABLENAME	String	File specification of the process's current command language interpreter (CLI) table.
TERMINAL	String	Login terminal name for interactive users (1–7 characters).
TMBU	Integer	Termination mailbox unit number.
TQCNT	Integer	Remaining timer queue entry quota.
TQLM	Integer	Timer queue entry quota.
UAF_FLAGS	Integer	User authorization file (UAF) flags from the UAF record of the user who owns the process.
UIC	String	Process's user identification code (UIC).
USERNAME	String	User name string (12 characters filled with trailing blanks).
VIRTPEAK	Integer	Peak virtual address size.
VOLUMES	Integer	Count of currently mounted volumes.
VP_CONSUMER	Boolean	Flag indicating whether the process is a vector consumer.
VP_CPUTIM	Integer	Total amount of time the process has accumulated as a vector customer.
WSAUTH	Integer	Maximum authorized working set size.
WSAUTHEXT	Integer	Maximum authorized working set extent.
WSEXTENT	Integer	Current working set extent.
WSPEAK	Integer	Working set peak.
WSQUOTA	Integer	Working set size quota.
WSSIZE	Integer	Process's current working set size.

If you use the F$GETJPI function to request information on the null process or the swapper process, you can specify any of the items in Table DCL1–9 except the following:

- ACCOUNT
- BYTLM
- ENQCNT
- ENQLM
- EXCVEC
- FILCNT
- FILM
- FINALEXC
- IMAGNAME
- LOGINTIM
- MSGMASK
- PAGFILCNT

- PGFLQUOTA
- PRCCNT
- PRCLM
- PROCPRIV
- SITESPEC
- TQCNT
- TQLM
- USERNAME
- VIRTPEAK
- VOLUMES
- WSPEAK

EXAMPLE

```
$  NAME = F$GETJPI("3B0018","USERNAME")
$  SHOW SYMBOL NAME
  NAME = "JANE"
```

This example shows how to use the F$GETJPI function to return the user name for the process number 3B0018. The user name is assigned to the symbol NAME.

F$GETQUI

Returns information about queues, batch and print jobs currently in those queues, form definitions, and characteristic definitions kept in the system job queue file.

Requires read (R) access to the job or SYSPRV (system privilege) or OPER (operator) privilege to obtain job and file information.

FORMAT

F$GETQUI(function,[item],[object-id],[flags])

return value

Either an integer or a character string, depending on the item you request. For items that return a Boolean value, the string is TRUE or FALSE. If the $GETQUI system service returns an error code, F$GETQUI returns a null string ("").

ARGUMENTS

function
Specifies the action that the F$GETQUI lexical function is to perform. F$GETQUI supports all functions that can be specified with the $GETQUI system service. The following table lists these functions:

Function	Description
CANCEL_OPERATION	Terminates any wildcard operation that may have been initiated by a previous call to F$GETQUI.
DISPLAY_CHARACTERISTIC	Returns information about a specific characteristic definition or the next characteristic definition in a wildcard operation.
DISPLAY_ENTRY	Returns information about a specific job entry or the next job entry that matches the selection criteria in a wildcard operation. The DISPLAY_ENTRY function code is similar to the DISPLAY_JOB function code in that both return job information. DISPLAY_JOB, however, requires that a call be made to establish queue context; DISPLAY_ENTRY does not require that queue context be established.
DISPLAY_FILE	Returns information about the next file defined for the current job context. Before you make a call to F$GETQUI to request file information, you must make a call to display queue and job information (with the DISPLAY_QUEUE and DISPLAY_JOB function codes) or to display entry information (with the DISPLAY_ENTRY function code).
DISPLAY_FORM	Returns information about a specific form definition or the next form definition in a wildcard operation.

Function	Description
DISPLAY_JOB	Returns information about the next job defined for the current queue context. Before you make a call to F$GETQUI to request job information, you must make a call to display queue information (with the DISPLAY_QUEUE function code). The DISPLAY_JOB function code is similar to the DISPLAY_ENTRY function code in that both return job information. DISPLAY_JOB, however, requires that a call be made to establish queue context; DISPLAY_ENTRY does not require that queue context be established.
DISPLAY_QUEUE	Returns information about a specific queue definition or the next queue definition in a wildcard operation.
TRANSLATE_QUEUE	Translates a logical name for a queue to the equivalence name for the queue.

Some function arguments cannot be specified with the **item-code,** the **object-id,** or the **flags** argument. The following table lists each function argument and corresponding format line to show whether the **item-code, object-id** and **flags** arguments are required, optional, or not applicable for that specific function. In the following format lines, brackets ([ ]) denote an optional argument. An omitted argument means the argument is not applicable for that function. Note that two commas (,,) must be used as placeholders to denote an omitted (whether optional or not applicable) argument.

Function	Format Line
CANCEL_OPERATION	F$GETQUI("CANCEL_OPERATION") or F$GETQUI(" ")
DISPLAY_CHARACTERISTIC	F$GETQUI("DISPLAY_CHARACTERISTIC",[item],object-id,[flags])
DISPLAY_ENTRY	F$GETQUI("DISPLAY_ENTRY",[item],[object-id],[flags])
DISPLAY_FILE	F$GETQUI("DISPLAY_FILE",[item],,[flags])
DISPLAY_FORM	F$GETQUI("DISPLAY_FORM",[item],object-id,[flags])
DISPLAY_JOB	F$GETQUI("DISPLAY_JOB",[item],,[flags])
DISPLAY_QUEUE	F$GETQUI("DISPLAY_QUEUE",[item],object-id,[flags])
TRANSLATE_QUEUE	F$GETQUI("TRANSLATE_QUEUE",[item],object-id)

item
Corresponds to a $GETQUI system service output item code. The **item** argument specifies the kind of information you want returned about a particular queue, job, file, form, or characteristic. Table DCL1–10 lists each item code and the data type of the value returned for each item code.

object-id
Corresponds to the $GETQUI system service QUI$SEARCH_NAME and QUI$_SEARCH_NUMBER input item codes. The **object-id** argument specifies either the name or the number of an object (for example, a specific queue name or form number) about which F$GETQUI is to return information. Wildcard characters (* and %) are allowed for the following functions:

- DISPLAY_CHARACTERISTIC
- DISPLAY_ENTRY

- DISPLAY_FORM
- DISPLAY_QUEUE

By specifying a wildcard as the **object-id** argument on successive calls, you can get status information about one or more jobs in a specific queue or about files within jobs in a specific queue. When a wildcard name is used, each call returns information for the next object (queue, form, and so on) in the list. A null string ("") is returned when the end of the list is reached. A wildcard can represent only object names, not object numbers.

flags
Specifies a list of keywords, separated by commas, that corresponds to the flags defined for the $GETQUI system service QUI$_SEARCH_FLAGS input item code. (These flags are used to define the scope of the object search specified in the call to the $GETQUI system service.) Note that the following keywords can be used only with certain function codes:

Keyword	Valid Function Code	Description
ALL_JOBS	DISPLAY_JOB	Requests that F$GETQUI search all jobs included in the established queue context. If you do not specify this flag, F$GETQUI returns information only about jobs that have the same user name as the caller.
BATCH	DISPLAY_QUEUE DISPLAY_ ENTRY	Selects batch queues.
EXECUTING_JOBS	DISPLAY_ENTRY DISPLAY_ JOB	Selects executing jobs.
FREEZE_CONTEXT	DISPLAY_ CHARACTERISTIC DISPLAY_ENTRY DISPLAY_FILE DISPLAY_FORM DISPLAY_JOB DISPLAY_QUEUE	When in wildcard mode, prevents advance of wildcard context to the next object. If you do not specify this flag, the context is advanced to the next object.
GENERIC	DISPLAY_QUEUE	Selects generic queues for searching.
HOLDING_JOBS	DISPLAY_ENTRY DISPLAY_ JOB	Selects jobs on unconditional hold.
PENDING_JOBS	DISPLAY_ENTRY DISPLAY_ JOB	Selects pending jobs.
PRINTER	DISPLAY_QUEUE DISPLAY_ ENTRY	Selects printer queues.
RETAINED_JOBS	DISPLAY_ENTRY DISPLAY_ JOB	Selects jobs being retained.
SERVER	DISPLAY_QUEUE DISPLAY_ ENTRY	Selects server queues.
SYMBIONT	DISPLAY_QUEUE DISPLAY_ ENTRY	Selects all output queues. Equivalent to specifying "PRINTER,SERVER,TERMINAL".
TERMINAL	DISPLAY_QUEUE DISPLAY_ ENTRY	Selects terminal queues.

Keyword	Valid Function Code	Description
THIS_JOB	DISPLAY_FILE DISPLAY_JOB DISPLAY_QUEUE	Selects all job file information about the calling batch job, the command file being executed, or the queue associated with the calling batch job.
TIMED_RELEASE_JOBS	DISPLAY_ENTRY DISPLAY_JOB	Selects jobs on hold until a specified time.
WILDCARD	DISPLAY_CHARACTERISTIC DISPLAY_ENTRY DISPLAY_FORM DISPLAY_QUEUE	Establishes and saves a context. Because the context is saved, the next operation can be performed based on that context.

DESCRIPTION

The F$GETQUI lexical function invokes the $GETQUI system service to return information about queues, batch and print jobs currently in those queues, form definitions, and characteristic definitions kept in the system job queue file. The F$GETQUI lexical function provides all the features of the $GETQUI system service, including wildcard and nested wildcard operations. For example, in nested wildcard operations, $GETQUI returns information about objects defined within another object. Specifically, this mode allows you to query jobs contained in a selected queue or files contained in a selected job in a sequence of calls. After each call, the system saves the GQC (internal GETQUI context block) so that the GQC can provide the queue or job context necessary for subsequent calls. For more information, see the description of the $GETQUI system service in the *VMS System Services Reference Manual*.

The F$GETQUI function returns information on all items that can be specified with the $GETQUI system service. Table DCL1–10 lists the items you can specify with the F$GETQUI function, the information returned, and the data type of this information.

Table DCL1–10   F$GETQUI Items

Item	Return Type	Information Returned
ACCOUNT_NAME	String	The account name of the owner of the specified job.
AFTER_TIME	String	The system time at or after which the specified job can execute.
ASSIGNED_QUEUE_NAME	String	The name of the execution queue to which the logical queue specified in the call to F$GETQUI is assigned.
BASE_PRIORITY	Integer	The priority at which batch jobs are initiated from a batch execution queue or the priority of a symbiont process that controls output execution queues.
CHARACTERISTICS	String	The characteristics associated with the specified queue or job.
CHARACTERISTIC_NAME	String	The name of the specified characteristic.
CHARACTERISTIC_NUMBER	Integer	The number of the specified characteristic.

Table DCL1-10 (Cont.)   F$GETQUI Items

Item	Return Type	Information Returned
CHECKPOINT_DATA	String	The value of the DCL symbol BATCH$RESTART when the specified batch job is restarted.
CLI	String	The name of the command language interpreter (CLI) used to execute the specified batch job. The file specification returned assumes the device name SYS$SYSTEM and the file type EXE.
COMPLETED_BLOCKS	Integer	The number of blocks that the symbiont has processed for the specified print job. This item code is applicable only to print jobs.
CONDITION_VECTOR	Integer	The completion status of the specified job.
CPU_DEFAULT	String	The default CPU time limit specified for the queue in delta time. This item code is applicable only to batch execution queues.
CPU_LIMIT	String	The maximum CPU time limit specified for the specified job or queue in delta time. This item code is applicable only to batch jobs and batch execution queues.
DEFAULT_FORM_NAME	String	The name of the default form associated with the specified output queue.
DEFAULT_FORM_STOCK	String	The name of the paper stock on which the specified default form is to be printed.
DEVICE_NAME	String	The node and device (or both) on which the specified execution queue is located. For output execution queues, only the device name is returned. The node name is used only in VAXcluster systems. The node name is specified by the SYSGEN parameter SCSNODE for the processor on which the queue executes.
		For batch execution queues, a null string ("") is returned. To get the name of the node on which a batch queue is executing, use the SCSNODE_NAME item.
ENTRY_NUMBER	Integer	The queue entry number of the specified job.
EXECUTING_JOB_COUNT	Integer	The number of jobs in the queue that are currently executing.
FILE_BURST	String	TRUE or FALSE to indicate whether burst and flag pages are to be printed preceding a file.
FILE_CHECKPOINTED	String	TRUE or FALSE to indicate whether the specified file is checkpointed.
FILE_COPIES	Integer	The number of times the specified file is to be processed. This item code is applicable only to output execution queues.
FILE_COPIES_DONE	Integer	The number of times the specified file has been processed. This item code is applicable only to output execution queues.
FILE_DELETE	String	TRUE or FALSE to indicate whether the specified file is to be deleted after execution of request.
FILE_DOUBLE_SPACE	String	TRUE or FALSE to indicate whether the symbiont formats the file with double spacing.
FILE_EXECUTING	String	TRUE or FALSE to indicate whether the specified file is being processed.

Table DCL1–10 (Cont.) F$GETQUI Items

Item	Return Type	Information Returned
FILE_FLAG	String	TRUE or FALSE to indicate whether a flag page is to be printed preceding a file.
FILE_FLAGS	Integer	The processing options that have been selected for the specified file. The integer represents a bit field. To find the settings of each bit in the field, use one of the following items in place of FILE_FLAGS: • FILE_BURST • FILE_PAGE_HEADER • FILE_DELETE • FILE_PAGINATE • FILE_DOUBLE_SPACE • FILE_PASSALL • FILE_FLAG • FILE_TRAILER
FILE_IDENTIFICATION	String	The internal file-identification value that uniquely identifies the selected file. This value specifies (in order) the following three file-identification fields in the RMS NAM block: • NAM$T_DVI (16 bytes) • NAM$W_FID (6 bytes) • NAM$W_DID (6 bytes)
FILE_PAGE_HEADER	String	TRUE or FALSE to indicate whether a page header is to be printed on each page of output.
FILE_PAGINATE	String	TRUE or FALSE to indicate whether the symbiont paginates output by inserting a form feed whenever output reaches the bottom margin of the form.
FILE_PASSALL	String	TRUE or FALSE to indicate whether the symbiont prints the file in PASSALL mode.
FILE_SETUP_MODULES	String	The names of the text modules that are to be extracted from the device control library and copied to the printer before the specified file is printed. This item code is meaningful only for output execution queues.
FILE_SPECIFICATION	String	The fully qualified RMS file specification of the file about which F$GETQUI is returning information.
FILE_STATUS	Integer	File status information. The integer represents a bit field. To find the settings of each bit in the field, use one of the following items in place of FILE_STATUS: • FILE_CHECKPOINTED • FILE_EXECUTING
FILE_TRAILER	String	TRUE or FALSE to indicate whether a trailer page is to be printed following a file.
FIRST_PAGE	Integer	The page number at which the printing of the specified file is to begin. This item code is applicable only to output execution queues.

Table DCL1–10 (Cont.) F$GETQUI Items

Item	Return Type	Information Returned
FORM_DESCRIPTION	String	The text string that describes the specified form to users and operators.
FORM_FLAGS	Integer	The processing options that have been selected for the specified form. The integer represents a bit field. To find the settings of each bit in the field, use one of the following items in place of FORM_FLAGS: • FORM_SHEET_FEED • FORM_TRUNCATE • FORM_WRAP
FORM_LENGTH	Integer	The physical length of the specified form in lines. This item code is applicable only to output execution queues.
FORM_MARGIN_BOTTOM	Integer	The bottom margin of the specified form in lines.
FORM_MARGIN_LEFT	Integer	The left margin of the specified form in characters.
FORM_MARGIN_RIGHT	Integer	The right margin of the specified form in characters.
FORM_MARGIN_TOP	Integer	The top margin of the specified form in lines.
FORM_NAME	String	The name of the specified form or the mounted form associated with the specified job or queue.
FORM_NUMBER	Integer	The number of the specified form.
FORM_SETUP_MODULES	String	The names of the text modules that are to be extracted from the device control library and copied to the printer before a file is printed on the specified form. This item code is meaningful only for output execution queues.
FORM_SHEET_FEED	String	TRUE or FALSE to indicate whether the symbiont pauses at the end of each physical page so that another sheet of paper can be inserted.
FORM_STOCK	String	The name of the paper stock on which the specified form is to be printed.
FORM_TRUNCATE	String	TRUE or FALSE to indicate whether the printer discards any characters that exceed the specified right margin.
FORM_WIDTH	Integer	The width of the specified form.
FORM_WRAP	String	TRUE or FALSE to indicate whether the printer prints any characters that exceed the specified right margin on the following line.
GENERIC_TARGET	String	The names of the execution queues that are enabled to accept work from the specified generic queue. This item code is meaningful only for generic queues.
HOLDING_JOB_COUNT	Integer	The number of jobs in the queue being held until explicitly released.
INTERVENING_BLOCKS	Integer	The number of blocks to be processed before the specified job can begin to execute. This item code is meaningful only for output execution queues.

Table DCL1–10 (Cont.)   F$GETQUI Items

Item	Return Type	Information Returned
INTERVENING_JOBS	Integer	The number of jobs that are to be processed before the specified job can begin to execute. This item code is meaningful only for output execution queues.
JOB_ABORTING	String	TRUE or FALSE to indicate whether the system is attempting to abort the execution of a job.
JOB_COPIES	Integer	The number of times the specified print job is to be repeated.
JOB_COPIES_DONE	Integer	The number of times that the specified print job has been repeated.
JOB_CPU_LIMIT	String	TRUE or FALSE to indicate whether a CPU time limit is specified for the job.
JOB_EXECUTING	String	TRUE or FALSE to indicate whether the specified job is executing or printing.
JOB_FILE_BURST	String	TRUE or FALSE to indicate whether a burst page option is explicitly specified for the job.
JOB_FILE_BURST_ONE	String	TRUE or FALSE to indicate whether burst and flag pages precede only the first copy of the first file in the job.
JOB_FILE_FLAG	String	TRUE or FALSE to indicate whether a flag page precedes each file in the job.
JOB_FILE_FLAG_ONE	String	TRUE or FALSE to indicate whether a flag page precedes only the first copy of the first file in the job.
JOB_FILE_PAGINATE	String	TRUE or FALSE to indicate whether a paginate option is explicitly specified for the job.
JOB_FILE_TRAILER	String	TRUE or FALSE to indicate whether a trailer page follows each file in the job.
JOB_FILE_TRAILER_ONE	String	TRUE or FALSE to indicate whether a trailer page follows only the last copy of the last file in the job.
JOB_FLAGS	Integer	The processing options selected for the specified job. The integer represents a bit field. To find the settings of each bit in the field, use one of the following items in place of JOB_FLAGS:

- JOB_CPU_LIMIT
- JOB_FILE_BURST
- JOB_FILE_BURST_ONE
- JOB_FILE_FLAG
- JOB_FILE_FLAG_ONE
- JOB_FILE_PAGINATE
- JOB_FILE_TRAILER
- JOB_FILE_TRAILER_ONE
- JOB_LOG_DELETE

- JOB_LOG_NULL
- JOB_LOG_SPOOL
- JOB_LOWERCASE
- JOB_NOTIFY
- JOB_RESTART
- JOB_WSDEFAULT
- JOB_WSEXTENT
- JOB_WSQUOTA

Table DCL1–10 (Cont.)   F$GETQUI Items

Item	Return Type	Information Returned
JOB_HOLDING	String	TRUE or FALSE to indicate whether the job will be held until it is explicitly released.
JOB_INACCESSIBLE	String	TRUE or FALSE to indicate whether the caller does not have read access to the specific job and file information in the system queue file. When FALSE, the DISPLAY_JOB and DISPLAY_FILE operations can return information for only the following output value item codes: • AFTER_TIME • COMPLETED_BLOCKS • ENTRY_NUMBER • INTERVENING_BLOCKS • INTERVENING_JOBS • JOB_SIZE • JOB_STATUS
JOB_LIMIT	Integer	The number of jobs that can execute simultaneously on the specified queue. This item code is applicable only to batch execution queues.
JOB_LOG_DELETE	String	TRUE or FALSE to indicate whether the log file is deleted after it is printed.
JOB_LOG_NULL	String	TRUE or FALSE to indicate whether a log file is not created.
JOB_LOG_SPOOL	String	TRUE or FALSE to indicate whether the job log file is queued for printing when the job is complete.
JOB_LOWERCASE	String	TRUE or FALSE to indicate whether the job is to be printed on a printer that can print both uppercase and lowercase letters.
JOB_NAME	String	The name of the specified job.
JOB_NOTIFY	String	TRUE or FALSE to indicate whether a message is broadcast to a terminal when a job completes or aborts.
JOB_PENDING	String	TRUE or FALSE to indicate whether the job is pending.
JOB_PID	String	The process identification (PID) number of the executing batch job.
JOB_REFUSED	String	TRUE or FALSE to indicate whether the job was refused by the symbiont and is waiting for the symbiont to accept it for processing.
JOB_RESET_MODULES	String	The names of the text modules that are to be extracted from the device control library and copied to the printer before each job in the specified queue is printed. This item code is meaningful only for output execution queues.
JOB_RESTART	String	TRUE or FALSE to indicate whether the job will restart after a system failure or can be requeued during execution.
JOB_RETAINED	String	TRUE or FALSE to indicate whether the job has completed, but is being retained in the queue.

Table DCL1–10 (Cont.)   F$GETQUI Items

Item	Return Type	Information Returned
JOB_SIZE	Integer	The total number of blocks in the specified print job.
JOB_SIZE_MAXIMUM	Integer	The maximum number of blocks that a print job initiated from the specified queue can contain. This item code is applicable only to output execution queues.
JOB_SIZE_MINIMUM	Integer	The minimum number of blocks that a print job initiated from the specified queue can contain. This item code is applicable only to output execution queues.
JOB_STARTING	String	TRUE or FALSE to indicate whether the job controller is starting to process the job and has begun communicating with an output symbiont or a job controller on another node.
JOB_STATUS	Integer	The specified job's status flags. The integer represents a bit field. To find the settings of each bit in the field, use one of the following items in place of JOB_STATUS: • ABORTING • JOB_REFUSED • JOB_EXECUTING • JOB_RETAINED • JOB_HOLDING • JOB_STARTING • JOB_INACCESSIBLE • JOB_SUSPENDED • JOB_PENDING • JOB_TIMED_RELEASE
JOB_SUSPENDED	String	TRUE or FALSE to indicate whether the job is suspended.
JOB_TIMED_RELEASE	String	TRUE or FALSE to indicate whether the job is waiting for a specified time to execute.
JOB_WSDEFAULT	String	TRUE or FALSE to indicate whether a default working set size is specified for the job.
JOB_WSEXTENT	String	TRUE or FALSE to indicate whether a working set extent is specified for the job.
JOB_WSQUOTA	String	TRUE or FALSE to indicate whether a working set quota is specified for the job.
LAST_PAGE	Integer	The page number at which the printing of the specified file should end. This item code is applicable only to output execution queues.
LIBRARY_SPECIFICATION	String	The name of the device control library for the specified queue. The library specification assumes the device and directory name SYS$LIBRARY and a file type of TLB. This item code is meaningful only for output execution queues.
LOG_QUEUE	String	The name of the queue into which the log file produced for the specified batch job is to be entered for printing. This item code is applicable only to batch jobs.
LOG_SPECIFICATION	String	The name of the log file to be produced for the specified job. This item code is meaningful only for batch jobs.

Table DCL1–10 (Cont.) F$GETQUI Items

Item	Return Type	Information Returned
NOTE	String	The note that is to be printed on the job flag and file flag pages of the specified job. This item code is meaningful only for output execution queues.
OPERATOR_REQUEST	String	The message that is to be sent to the queue operator before the specified job begins to execute. This item code is meaningful only for output execution queues.
OWNER_UIC	String	The owner user identification code (UIC) of the specified queue.
PAGE_SETUP_MODULES	String	The names of the text modules to be extracted from the device control library and copied to the printer before each page of the specified form is printed.
PARAMETER_1 to PARAMETER_8	String	The value of the user-defined parameters that become the value of the DCL symbols P1 to P8 respectively.
PENDING_JOB_BLOCK_ COUNT	Integer	The total number of blocks for all pending jobs in the queue (valid only for output execution queues).
PENDING_JOB_COUNT	Integer	The number of jobs in the queue in a pending state.
PENDING_JOB_REASON	Integer	The reason that the job is in a pending state. The integer represents a bit field. To find the settings of each bit in the field, use one of the following items in place of PENDING_JOB_ REASON: • PEND_CHAR_MISMATCH • PEND_NO_ACCESS • PEND_JOB_SIZE_MAX • PEND_QUEUE_BUSY • PEND_JOB_SIZE_MIN • PEND_QUEUE_STATE • PEND_LOWERCASE_ MISMATCH • PEND_STOCK_ MISMATCH
PEND_CHAR_MISMATCH	String	TRUE or FALSE to indicate whether the job requires characteristics that are not available on the execution queue.
PEND_JOB_SIZE_MAX	String	TRUE or FALSE to indicate whether the block size of the job exceeds the upper block limit of the execution queue.
PEND_JOB_SIZE_MIN	String	TRUE or FALSE to indicate whether the block size of the job is less than the lower limit of the execution queue.
PEND_LOWERCASE_ MISMATCH	String	TRUE or FALSE to indicate whether the job requires a lowercase printer.
PEND_NO_ACCESS	String	TRUE or FALSE to indicate whether the owner of the job does not have access to the execution queue.
PEND_QUEUE_BUSY	String	TRUE or FALSE to indicate whether the job is pending because the number of jobs currently executing on the queue equals the job limit for the queue.
PEND_QUEUE_STATE	String	TRUE or FALSE to indicate whether the job is pending because the execution queue is not in a running open state.

Table DCL1-10 (Cont.)   F$GETQUI Items

Item	Return Type	Information Returned
PEND_STOCK_MISMATCH	String	TRUE or FALSE to indicate whether the stock type required by the job's form does not match the stock type of the form mounted on the execution queue.
PRIORITY	Integer	The scheduling priority of the specified job.
PROCESSOR	String	The name of the symbiont image that executes print jobs initiated from the specified queue.
PROTECTION	String	The specified queue's protection mask.
QUEUE_ACL_SPECIFIED	String	TRUE or FALSE to indicate whether an access control list has been specified for the queue.
QUEUE_ALIGNING	String	TRUE or FALSE to indicate whether the queue is currently printing alignment pages. A queue prints alignment pages when it is restarted from a paused state by using the command START /QUEUE/ALIGN.
QUEUE_BATCH	String	TRUE or FALSE to indicate whether the queue is a batch queue or a generic batch queue.
QUEUE_CLOSED	String	TRUE or FALSE to indicate whether the queue is closed and will not accept new jobs until the queue is put in an open state.
QUEUE_CPU_DEFAULT	String	TRUE or FALSE to indicate whether a default CPU time limit has been specified for all jobs in the queue.
QUEUE_CPU_LIMIT	String	TRUE or FALSE to indicate whether a maximum CPU time limit has been specified for all jobs in the queue.
QUEUE_DESCRIPTION	String	The description of the queue that was defined by using the /DESCRIPTION qualifier with the INITIALIZE/QUEUE command.
QUEUE_FILE_BURST	String	TRUE or FALSE to indicate whether burst and flag pages precede each file in each job initiated from the queue.
QUEUE_FILE_BURST_ONE	String	TRUE or FALSE to indicate whether burst and flag pages precede only the first copy of the first file in each job initiated from the queue.
QUEUE_FILE_FLAG	String	TRUE or FALSE to indicate whether a flag page precedes each file in each job initiated from the queue.
QUEUE_FILE_FLAG_ONE	String	TRUE or FALSE to indicate whether a flag page precedes only the first copy of the first file in each job initiated from the queue.
QUEUE_FILE_PAGINATE	String	TRUE or FALSE to indicate whether the output symbiont paginates output for each job initiated from this queue. The output symbiont paginates output by inserting a form feed whenever output reaches the bottom margin of the form.
QUEUE_FILE_TRAILER	String	TRUE or FALSE to indicate whether a trailer page follows each file in each job initiated from the queue.
QUEUE_FILE_TRAILER_ONE	String	TRUE or FALSE to indicate whether a trailer page follows only the last copy of the last file in each job initiated from the queue.

Table DCL1–10 (Cont.)   F$GETQUI Items

Item	Return Type	Information Returned
QUEUE_FLAGS	Integer	The processing options that have been selected for the specified queue. The integer represents a bit field. To find the settings of each bit in the field, use one of the following items in place of QUEUE_FLAGS:

- QUEUE_ACL_SPECIFIED
- QUEUE_BATCH
- QUEUE_CPU_DEFAULT
- QUEUE_CPU_LIMIT
- QUEUE_FILE_BURST
- QUEUE_FILE_BURST_ ONE
- QUEUE_FILE_FLAG
- QUEUE_FILE_FLAG_ONE
- QUEUE_FILE_PAGINATE
- QUEUE_FILE_TRAILER
- QUEUE_FILE_TRAILER_ ONE
- QUEUE_GENERIC
- QUEUE_GENERIC_ SELECTION

- QUEUE_JOB_BURST
- QUEUE_JOB_FLAG
- QUEUE_JOB_SIZE_ SCHED
- QUEUE_JOB_TRAILER
- QUEUE_PRINTER
- QUEUE_RECORD_ BLOCKING
- QUEUE_RETAIN_ALL
- QUEUE_RETAIN_ ERROR
- QUEUE_SWAP
- QUEUE_TERMINAL
- QUEUE_WSDEFAULT
- QUEUE_WSEXTENT
- QUEUE_WSQUOTA

Item	Return Type	Information Returned
QUEUE_GENERIC	String	TRUE or FALSE to indicate whether the queue is a generic queue.
QUEUE_GENERIC_ SELECTION	String	TRUE or FALSE to indicate whether the queue is an execution queue that can accept work from a generic queue.
QUEUE_IDLE	String	TRUE or FALSE to indicate whether the queue contains no job requests.
QUEUE_JOB_BURST	String	TRUE or FALSE to indicate whether burst and flag pages precede each job initiated from the queue.
QUEUE_JOB_FLAG	String	TRUE or FALSE to indicate whether a flag page precedes each job initiated from the queue.
QUEUE_JOB_SIZE_SCHED	String	TRUE or FALSE to indicate whether jobs initiated from the queue are scheduled according to size with the smallest job of a given priority processed first. (Meaningful only for output queues.)
QUEUE_JOB_TRAILER	String	TRUE or FALSE to indicate whether a trailer page follows each job initiated from the queue.
QUEUE_LOWERCASE	String	TRUE or FALSE to indicate whether queue is associated with a printer that can print both uppercase and lowercase characters.
QUEUE_NAME	String	The name of the specified queue or the name of the queue that contains the specified job.

Table DCL1–10 (Cont.)   F$GETQUI Items

Item	Return Type	Information Returned
QUEUE_PAUSED	String	TRUE or FALSE to indicate whether execution of all current jobs in the queue is temporarily halted.
QUEUE_PAUSING	String	TRUE or FALSE to indicate whether the queue is temporarily halting execution. Currently executing jobs are completing; temporarily, no new jobs can begin executing.
QUEUE_PRINTER	String	TRUE or FALSE to indicate whether the queue is a printer queue.
QUEUE_RECORD_BLOCKING	String	TRUE or FALSE to indicate whether the symbiont is permitted to concatenate, or block together, the output records it sends to the output device.
QUEUE_REMOTE	String	TRUE or FALSE to indicate whether the queue is assigned to a physical device that is not connected to the local node.
QUEUE_RESETTING	String	TRUE or FALSE to indicate whether the queue is resetting and stopping.
QUEUE_RESUMING	String	TRUE or FALSE to indicate whether the queue is restarting after pausing.
QUEUE_RETAIN_ALL	String	TRUE or FALSE to indicate whether all jobs initiated from the queue remain in the queue after they finish executing. Completed jobs are marked with a completion status.
QUEUE_RETAIN_ERROR	String	TRUE or FALSE to indicate whether only jobs that do not complete successfully are retained in the queue.
QUEUE_SERVER	String	TRUE or FALSE to indicate whether queue processing is directed to a server symbiont.
QUEUE_STALLED	String	TRUE or FALSE to indicate whether the physical device to which the queue is assigned is stalled; that is, the device has not completed the last I/O request submitted to it.
QUEUE_STARTING	String	TRUE or FALSE to indicate whether the queue is starting.
QUEUE_STATUS	Integer	The specified queue's status flags. The integer represents a bit field. To find the settings of each bit in the field, use one of the following items in place of QUEUE_STATUS:

• QUEUE_ALIGNING	• QUEUE_RESUMING
• QUEUE_CLOSED	• QUEUE_SERVER
• QUEUE_IDLE	• QUEUE_STALLED
• QUEUE_LOWERCASE	• QUEUE_STARTING
• QUEUE_PAUSED	• QUEUE_STOPPED
• QUEUE_PAUSING	• QUEUE_STOPPING
• QUEUE_REMOTE	• QUEUE_UNAVAILABLE
• QUEUE_RESETTING	

Item	Return Type	Information Returned
QUEUE_STOPPED	String	TRUE or FALSE to indicate whether the queue is stopped.
QUEUE_STOPPING	String	TRUE or FALSE to indicate whether the queue is stopping.

Table DCL1–10 (Cont.)   F$GETQUI Items

Item	Return Type	Information Returned
QUEUE_SWAP	String	TRUE or FALSE to indicate whether jobs initiated from the queue can be swapped.
QUEUE_TERMINAL	String	TRUE or FALSE to indicate whether the queue is a terminal queue.
QUEUE_UNAVAILABLE	String	TRUE or FALSE to indicate whether the physical device to which queue is assigned is not available.
QUEUE_WSDEFAULT	String	TRUE or FALSE to indicate whether a default working set size is specified for each job initiated from the queue.
QUEUE_WSEXTENT	String	TRUE or FALSE to indicate whether a working set extent is specified for each job initiated from the queue.
QUEUE_WSQUOTA	String	TRUE or FALSE to indicate whether a working set quota is specified for each job initiated from the queue.
REQUEUE_QUEUE_NAME	String	The name of the queue to which the specified job is reassigned.
RESTART_QUEUE_NAME	String	The name of the queue in which the job will be placed if the job is restarted.
RETAINED_JOB_COUNT	Integer	The number of jobs in the queue retained after successful completion plus those retained on error.
SCSNODE_NAME	String	The 6-byte name of the VAX node on which jobs initiated from the specified queue execute. The node name matches the value of the SYSGEN parameter SCSNODE for the target node.
SUBMISSION_TIME	String	The time at which the specified job was submitted to the queue.
TIMED_RELEASE_JOB_ COUNT	Integer	The number of jobs in the queue on hold until a specified time.
UIC	String	The user identification code (UIC) of the owner of the specified job.
USERNAME	String	The user name of the owner of the specified job.
WSDEFAULT	Integer	The default working set size specified for the specified job or queue. This value is meaningful only for batch jobs and execution and output queues.
WSEXTENT	Integer	The working set extent specified for the specified job or queue. This value is meaningful only for batch jobs and execution and output queues.
WSQUOTA	Integer	The working set quota for the specified job or queue. This value is meaningful only for batch jobs and execution and output queues.

EXAMPLES

1   `$  BLOCKS = F$GETQUI("DISPLAY_ENTRY" "JOB_SIZE", 1347)`

> In this example, the F$GETQUI lexical function is used to obtain the size in blocks of print job 1347. The value returned reflects the total number of blocks occupied by the files associated with the job.

2   `$ IF F$GETQUI("DISPLAY_QUEUE", "QUEUE_STOPPED", "VAX1_BATCH") .EQS.`
`"TRUE" THEN GOTO 500`

> In this example, the F$GETQUI lexical function is used to return a value
> of TRUE or FALSE depending on whether the queue VAX1_BATCH is in a
> stopped state. If VAX1_BATCH is not in the system, F$GETQUI returns a
> null string ("").

3

```
! This command procedure shows all queues and the jobs in them.
$   TEMP = F$GETQUI("")
$   QLOOP:
$   QNAME = F$GETQUI("DISPLAY_QUEUE","QUEUE_NAME","*")
$   IF QNAME .EQS. "" THEN EXIT
$   WRITE SYS$OUTPUT ""
$   WRITE SYS$OUTPUT "QUEUE: ", QNAME
$   JLOOP:
$   NOACCESS = F$GETQUI("DISPLAY_JOB","JOB_INACCESSIBLE",,"ALL_JOBS")
$   IF NOACCESS .EQS. "TRUE" THEN GOTO JLOOP
$   IF NOACCESS .EQS. "" THEN GOTO QLOOP
$   JNAME = F$GETQUI("DISPLAY_JOB","JOB_NAME",,"FREEZE_CONTEXT,ALL_JOBS")
$   WRITE SYS$OUTPUT "    JOB:  ", JNAME
$   GOTO JLOOP
```

> This sample command procedure displays all the queues in the system and
> all the jobs to which the user has read access in the system. In the outer
> loop a wildcard display queue operation is performed. No call is made
> to establish the right to obtain information about the queue, because all
> users have implicit read access to queue attributes. Because a wildcard
> queue name is specified ("*"), wildcard queue context is maintained across
> calls to F$GETQUI.
>
> In the inner loop, to obtain information about all jobs, we enter nested
> wildcard mode from wildcard display queue mode. In this loop, a call is
> made to establish the right to obtain information about these jobs because
> users do not have implicit read access to jobs. The FREEZE_CONTEXT
> keyword is used in the request for a job name to prevent the advance
> of the wildcard context to the next object. After the job name has been
> retrieved and displayed, the procedure loops back up for the next job. The
> context is advanced because the procedure has not used the FREEZE_
> CONTEXT keyword. The wildcard queue context is dissolved when the list
> of matching queues is exhausted. Finally, F$GETQUI returns a null string
> ("") to denote that no more objects match the specified search criteria.

F$GETSYI

Returns status and identification information about the local system (or about a node in the local VAXcluster, if your system is part of a VAXcluster).

FORMAT	**F$GETSYI***(item [,node-name] [,cluster-id])*

return value   Either an integer or a character string, depending on the item you request.

ARGUMENTS   *item*
Indicates the type of information to be reported about the local node (or about another node in your VAXcluster, if your system is part of a VAXcluster). Specify the item as a character string expression.

You can specify the items in Table DCL1–11 only for your local node; you cannot specify the node argument with these items. You can specify these items whether or not you are in a VAXcluster.

You can specify the items in Table DCL1–12 for either your local node or for another node in your VAXcluster. The information in this table is returned for your local node if you do not specify the **node-name** argument; the information is returned for the specified node if you include the **node-name** argument. Your system must be a member of a VAXcluster in order to specify the items in this table, except for CLUSTER_MEMBER. You can specify CLUSTER_MEMBER whether or not your system is a member of a VAXcluster.

You can also specify any of the SYSGEN parameters listed in the *VMS System Generation Utility Manual*. However, you can specify SYSGEN parameters only for your local node; you cannot specify the **node-name** argument with these items.

node-name
Specifies the node in your VAXcluster for which information is to be returned. Specify the node as a character string expression. You cannot use wildcards to specify the **node-name** argument. (This argument can be specified only if your system is part of a VAXcluster.)

You can request information about another node in your VAXcluster only when you specify an item from Table DCL1–12. If you do not specify a node, the default is the current node.

cluster-id
Specifies the cluster node identification number for which the information is to be returned. (This argument can be specified only if your system is part of a VAXcluster.)

You can request information about another node in your VAXcluster only when you specify an item from Table DCL1–12. If you do not specify a node, the default is the current node.

To get information for all the nodes in a cluster, use the F$CSID lexical function to obtain each cluster system identification number and use the **cluster-id** argument of F$GETSYI to gather information about each node.

DESCRIPTION

The F$GETSYI lexical function invokes the $GETSYI system service to return status and identification information about the local system (or about a node in the local VAXcluster, if your system is part of a VAXcluster). The F$GETSYI function returns information on the items that can be specified with the $GETSYI system service. For more information on the $GETSYI system service, see the *VMS System Services Reference Manual*.

You can specify the node for which you want information by supplying either the **node-name** or the **cluster-id** argument, but not both.

Table DCL1–11 lists the items you can specify with the F$GETSYI lexical function to get information about your local node. Table DCL1–12 lists the items you can specify to get information about either your local node or another node in your VAXcluster.

Table DCL1–11   F$GETSYI Items for the Local Node Only

Item	Return Type	Information Returned
ACTIVECPU_CNT	Integer	The count of CPUs actively participating in the current boot of a symmetric multiprocessing (SMP) system.
ARCHFLAG	String	Architecture flags for the system.
AVAILCPU_CNT	Integer	The count of CPUs recognized in the system.
BOOTTIME	String	The time the system was booted.
CHARACTER_EMULATED	String	TRUE or FALSE to indicate whether the character string instructions are emulated on the CPU.
CONTIG_GBLPAGES	Integer	Total number of free, contiguous global pages.
CPU	Integer	The processor type, as represented in the processor's system identification (SID) register. For example, the integer 1 represents a VAX–11/780 and the integer 6 represents a VAX 8530, VAX 8550, VAX 8700, or VAX 8800.
DECIMAL_EMULATED	String	TRUE or FALSE to indicate whether the decimal string instructions are emulated on the CPU.
D_FLOAT_EMULATED	String	TRUE or FALSE to indicate whether the D_floating instructions are emulated on the CPU.

Table DCL1-11 (Cont.)   F$GETSYI Items for the Local Node Only

Item	Return Type	Information Returned
ERRORLOGBUFFERS	Integer	Number of system pages in use as buffers for error logging.
F_FLOAT_EMULATED	String	TRUE or FALSE to indicate whether the F_floating instructions are emulated on the CPU.
FREE_GBLPAGES	Integer	Current count of free global pages.
FREE_GBLSECTS	Integer	Current count of free global section table entries.
G_FLOAT_EMULATED	String	TRUE or FALSE to indicate whether the G_floating instructions are emulated on the CPU.
PAGEFILE_FREE	Integer	Number of free pages in the currently installed paging files.
PAGEFILE_PAGE	Integer	Number of pages in the currently installed paging files.
SID	Integer	System identification register.
SWAPFILE_FREE	Integer	Number of free pages in the currently installed swapping files.
SWAPFILE_PAGE	Integer	Number of pages in the currently installed swapping files.
SYSTEM_RIGHTS	String	The contents of the system rights list on the local system. If you specify a remote system, a null string (" ") is returned. This item code returns a list of identifier names separated by commas (,).
VERSION	String	Version of VMS in use (8-character string filled with trailing blanks).

Table DCL1-12   F$GETSYI Items for the Local Node or for Other Nodes in the VAXCluster

Item	Return Type	Information Returned
CLUSTER_FSYSID	String	System identification number for first node to boot in the VAXcluster (the founding node). This number is returned as a character string containing a hexadecimal number.
CLUSTER_FTIME	String	The time when the first node in the VAXcluster was booted.
CLUSTER_MEMBER	String	TRUE or FALSE if the node is a member of the local VAXcluster.

Table DCL1–12 (Cont.)   F$GETSYI Items for the Local Node or for Other Nodes in the VAXCluster

Item	Return Type	Information Returned
CLUSTER_NODES	Integer	Total number of nodes in the VAXcluster, as an integer.
CLUSTER_ QUORUM	Integer	Total quorum for the VAXcluster.
CLUSTER_VOTES	Integer	Total number of votes in the VAXcluster.
HW_MODEL	Integer	An integer that identifies the node's VAX model type.
HW_NAME	String	The VAX model name.
NODENAME	String	Node name (does not include the following double colon).
NODE_AREA	Integer	The VAX DECnet area for the target node.
NODE_CSID	String	The CSID of the specified node, as a string containing a hexadecimal number. The CSID is a form of system identification.
NODE_HWTYPE	String	Hardware type of the specified node.
NODE_HWVERS	String	Hardware version of the specified node.
NODE_NUMBER	Integer	The VAX DECnet number for the specified node.
NODE_QUORUM	Integer	Quorum that the node has.
NODE_SWINCARN	String	Software incarnation number for the specified node. This number is returned as a string containing a hexadecimal number.
NODE_SWTYPE	String	Type of operating system software used by the specified node.
NODE_SWVERS	String	Software version of the specified node.
NODE_SYSTEMID	String	System identification number for the specified node. This number is returned as a string containing a hexadecimal number.
NODE_VOTES	Integer	Number of votes allotted to the node.
SCS_EXISTS	String	TRUE or FALSE to indicate whether the system communication subsystem (SCS) is currently loaded on a VAX node.
VECTOR_ EMULATOR	Boolean	Flag indicating the presence of the VAX vector instruction emulator facility (VVIEF) in the system.
VP_MASK	Integer	Mask indicating which processors in the system have vector coprocessors.
VP_NUMBER	Integer	Number of vector processors in the system.

EXAMPLES

1
```
$ SYSID = F$GETSYI("SID")
$ SHOW SYMBOL SYSID
  SYSID = 19923201  Hex = 01300101 Octal = 000401
```

> This example shows how to use the F$GETSYI function to return the information in the system identification register. Use quotation marks (" ") around the argument SID because it is a string literal. The value returned by F$GETSYI is assigned to the symbol SYSID. Because a node is not specified, information about your current node is returned.

2
```
$ MEM = F$GETSYI("CLUSTER_MEMBER", "LONDON")
$ SHOW SYMBOL MEM
  MEM = "TRUE"
```

> This example uses the F$GETSYI function to determine whether the node LONDON is a member of the local VAXcluster. The return value TRUE indicates that the remote node LONDON is a member of the VAXcluster.

3
```
$ LIM = F$GETSYI("BJOBLIM")
$ SHOW SYMBOL LIM
  LIM = 16   Hex = 00000010  Octal = 00000000020
```

> This example uses the SYSGEN parameter BJOBLIM as an argument for the F$GETSYI function. This argument returns the batch job limit for the current system.

F$IDENTIFIER

Converts an alphanumeric identifier to its integer equivalent, or converts an integer identifier to its alphanumeric equivalent. An identifier is a name or number that identifies a category of users of a data resource. The system uses identifiers to determine a user's access to a resource.

FORMAT

F$IDENTIFIER(identifier,conversion-type)

return value

An integer value if you are converting an identifier from a name to an integer. The F$IDENTIFIER function returns a string if you are converting an identifier from an integer to a name. If you specify an identifier that is not valid, the F$IDENTIFIER function returns a null string (" ") (if you are converting from number to name) or a zero (if you are converting from name to number).

ARGUMENTS

identifier
Specifies the identifier to be converted. Specify the identifier as an integer expression if you are converting an integer to a name. Specify the identifier as a character string expression if you are converting a name to an integer.

conversion-type
Indicates the type of conversion to be performed. If the **identifier** argument is alphanumeric, specify the **conversion-type** argument as a character string containing "NAME_TO_NUMBER". If the **identifier** argument is numeric, specify the **conversion-type** argument as a character string containing "NUMBER_TO_NAME".

EXAMPLES

1
```
$ UIC_INT= F$IDENTIFIER("SLOANE","NAME_TO_NUMBER")
$ SHOW SYMBOL UIC_INT
  UIC_INT = 15728665   Hex = 00F00019  Octal = 00074000031
$ UIC = F$FAO("!%U",UIC_INT)
  SHOW SYMBOL UIC
  UIC = [360,031]
```

This example uses the F$IDENTIFIER to convert the member identifier from the UIC [MANAGERS,SLOANE] to an integer. The F$IDENTIFIER function shows that the member identifier SLOANE is equivalent to the integer 15728665. Note that you must specify the identifier SLOANE using uppercase letters.

To convert this octal number to a standard numeric user identification code (UIC), use the F$FAO function with the !%U directive. (This directive converts a longword to a UIC in named format.) In this example, the member identifier SLOANE is equivalent to the numeric UIC [360,031].

2
```
$  UIC_INT = (%O31 + (%X10000 * %O360))
$  UIC_NAME = F$IDENTIFIER(UIC_INT,"NUMBER_TO_NAME")
$  SHOW SYMBOL UIC_NAME
  UIC_NAME = "ODONNELL"
```

This example obtains the alphanumeric identifier associated with the numeric UIC [360,031]. First, you must obtain the longword integer that corresponds to the UIC [360,031]. To do this, place the member number into the low-order word. Place the group number into the high order word. Next, use the F$IDENTIFIER function to return the named identifier associated with the integer.

F$INTEGER

Returns the integer equivalent of the result of the specified expression.

FORMAT   **F$INTEGER***(expression)*

---

return value   An integer value that is equivalent to the specified expression.

ARGUMENT   *expression*
Specifies the expression to be evaluated. Specify either an integer or a character string expression.

If you specify an integer expression, the F$INTEGER function evaluates the expression and returns the result. If you specify a string expression, the F$INTEGER function evaluates the expression, converts the resulting string to an integer, and returns the result.

After evaluating a string expression, the F$INTEGER function converts the result to an integer in the following way. If the resulting string contains characters that form a valid integer, the F$INTEGER function returns the integer value. If the string contains characters that do not form a valid integer, the F$INTEGER function returns the integer 1 if the string begins with T, t, Y, or y. The function returns the integer 0 if the string begins with any other character.

EXAMPLE

```
$  A = "23"
$  B = F$INTEGER("-9" + A)
$  SHOW SYMBOL B
  B = -923 Hex=FFFFFC65 Octal=176145
```

This example shows how to use the F$INTEGER function to equate a symbol to the integer value returned by the function. In the example, the F$INTEGER function returns the integer equivalent of the string expression ("-9" + A). First, the F$INTEGER function evaluates the string expression by concatenating the string literal "-9" with the string literal "23". Note that the value of the symbol A is substituted automatically in a string expression. Also note that the plus sign (+) is a string concatenation operator since both arguments are string literals.

After the string expression is evaluated, the F$INTEGER function converts the resulting character string ("-923") to an integer, and returns the value -923. This integer value is assigned to the symbol B.

F$LENGTH

Returns the length of the specified character string.

FORMAT

F$LENGTH(string)

return value

An integer value for the length of the string.

ARGUMENT

string
Specifies the character string whose length is being determined. Specify the string argument as a character string expression.

EXAMPLE

```
$  MESSAGE = F$MESSAGE(%X1C)
$  SHOW SYMBOL MESSAGE
  MESSAGE = "%SYSTEM-F-EXQUOTA, exceeded quota"
$  STRING_LENGTH = F$LENGTH(MESSAGE)
$  SHOW SYMBOL STRING_LENGTH
  STRING_LENGTH = 33   Hex = 00000021  Octal = 000041
```

The first assignment statement uses the F$MESSAGE function to return the message that corresponds to the hexadecimal value 1C. The message is returned as a character string and is assigned to the symbol MESSAGE.

The F$LENGTH function is then used to return the length of the character string assigned to the symbol MESSAGE. You do not need to use quotation marks (" ") when you use the symbol MESSAGE as an argument for the F$LENGTH function. (Quotation marks are not used around symbols in character string expressions.)

The F$LENGTH function returns the length of the character string and assigns it to the symbol STRING_LENGTH. At the end of the example, the symbol STRING_LENGTH has a value equal to the number of characters in the value of the symbol named MESSAGE, that is, 33.

F$LOCATE

Locates a specified portion of a character string and returns as an integer the offset of the first character. If the substring is not found, F$LOCATE returns the length (the offset of the last character in the character string plus one) of the searched string.

FORMAT   **F$LOCATE**(*substring,string*)

return value

An integer value representing the offset of the substring argument. An offset is the position of a character or a substring relative to the beginning of the string. The first character in a string is always offset position 0 from the beginning of the string (which always begins at the leftmost character).

If the substring is not found, the F$LOCATE function returns an offset of the last character in the character string plus 1. (This equals the length of the string.)

ARGUMENTS   *substring*

Specifies the character string that you want to locate within the string specified in the **string** argument.

string

Specifies the character string to be edited by F$LOCATE.

EXAMPLES

1
```
$ FILE_SPEC = "MYFILE.DAT;1"
$ NAME_LENGTH = F$LOCATE(".",FILE_SPEC)
```

The F$LOCATE function in this example returns the position of the period (.) in the string with respect to the beginning of the string. The period is in offset position 6, so the value 6 is assigned to the symbol NAME_LENGTH. Note that NAME_LENGTH also equals the length of the file name portion of the file specification MYFILE.DAT, that is, 6.

The substring argument, the period, is specified as a string literal and is therefore enclosed in quotation marks (" "). The **string** argument FILE_SPEC is a symbol, so it should not be placed within quotation marks. It is automatically replaced by its current value during the processing of the function.

```
2   $ INQUIRE TIME "Enter time"
    $ IF F$LOCATE(":",TIME) .EQ. F$LENGTH(TIME) THEN -
    GOTO NO_COLON
```

This section of a command procedure compares the results of the F$LOCATE and F$LENGTH functions to see if they are equal. This technique is commonly used to determine whether a character or substring is contained in a string.

In the example, the INQUIRE command prompts for a time value and assigns the user-supplied time to the symbol TIME. The IF command checks for the presence of a colon (:) in the string entered in response to the prompt. If the value returned by the F$LOCATE function equals the value returned by the F$LENGTH function, the colon is not present. You use the .EQ. operator (rather than .EQS.) because the F$LOCATE and F$LENGTH functions return integer values.

Note that quotation marks are used around the substring argument, the colon, because it is a string literal. However, the symbol TIME does not require quotation marks because it is automatically evaluated as a string expression.

F$MESSAGE

Returns as a character string the facility, severity, identification, and text associated with the specified system status code.

FORMAT   **F$MESSAGE***(status-code)*

---

return value   A character string containing the system message that corresponds to the argument you specify.

Note that, although each message in the system message file has a numeric value or range of values associated with it, there are many possible numeric values that do not have corresponding messages. If you specify an argument that has no corresponding message, the F$MESSAGE function returns a string containing the NOMSG error message.

For more information on system error messages, see the *VMS System Messages and Recovery Procedures Reference Manual.*

ARGUMENT   ***status-code***
Specifies the status code for which you are requesting error message text. You must specify the status code as an integer expression.

EXAMPLE

```
$  ERROR_TEXT = F$MESSAGE(%X1C)
$  SHOW SYMBOL ERROR_TEXT
   ERROR_TEXT = "%SYSTEM-F-EXQUOTA, exceeded quota"
```

This example shows how to use the F$MESSAGE function to determine the message associated with the status code %X1C. The F$MESSAGE function returns the message string, which is assigned to the symbol ERROR_TEXT.

F$MODE

Returns a character string showing the mode in which a process is executing. The F$MODE function has no arguments, but must be followed by parentheses.

FORMAT   **F$MODE( )**

return value

The character string INTERACTIVE for interactive processes. If the process is noninteractive, the character string BATCH, NETWORK or OTHER is returned. Note that the return string always contains uppercase letters.

ARGUMENTS   *None.*

DESCRIPTION

The lexical function F$MODE returns a character string showing the mode in which a process is executing. The F$MODE function has no arguments, but must be followed by parentheses.

The F$MODE function is useful in command procedures that must operate differently when executed interactively and noninteractively. You should include either the F$MODE function or the F$ENVIRONMENT function in your login command file to execute different commands for interactive terminal sessions and noninteractive sessions.

If you do not include the F$MODE function to test whether your login command file is being executed from an interactive process, and the login command file is executed from a noninteractive process (such as a batch job), the process may terminate if the login command file contains commands that are appropriate only for interactive processing.

A command procedure can use the F$MODE function to test whether the procedure is being executed during an interactive terminal session. It can direct the flow of execution according to the results of this test.

EXAMPLE

```
$ IF F$MODE() .NES. "INTERACTIVE" THEN GOTO NON_INT_DEF
$ INTDEF:              ! Commands for interactive terminal sessions
   .
   .
   .
$ EXIT
$ NON_INT_DEF:           !Commands for noninteractive processes
   .
   .
   .
```

This example shows the beginning of a login.com file that has two
sets of initialization commands: one for interactive mode and one for
noninteractive mode (including batch and network jobs). The IF command
compares the character string returned by F$MODE with the character
string INTERACTIVE; if they are not equal, control branches to the
label NON_INT_DEF. If the character strings are equal, the statements
following the label INTDEF are executed and the procedure exits before
the statements at NON_INT_DEF.

F$PARSE

Parses a file specification and returns either the expanded file specification or the particular file specification field that you request.

FORMAT

F$PARSE(filespec [,default-spec] [,related-spec] [,field] [,parse-type])

return value

A character string containing the expanded file specification or the field you specify. If you do not provide a complete file specification for the **filespec** argument, the F$PARSE function supplies defaults in the return string, as described in the Description section.

If an error is detected during the parse, the F$PARSE function returns a null string (""), except when you specify a field name or the SYNTAX_ ONLY parse type.

ARGUMENTS

filespec
Specifies a character string containing the file specification to be parsed.

The file specification can contain wildcard characters (* and %). If you use a wildcard character, the file specification returned by the F$PARSE function contains the wildcard.

default-spec
Specifies a character string containing the default file specification.

The fields in the default file specification are substituted in the output string if a particular field in the **filespec** argument is missing. You can make further substitutions in the **filespec** argument by using the **related-spec** argument.

related-spec
Specifies a character string containing the related file specification.

The fields in the related file specification are substituted in the output string if a particular field is missing from both the **filespec** and **default-spec** arguments.

field
Specifies a character string containing the name of a field in a file specification. Specifying the **field** argument causes the F$PARSE function to return a specific portion of a file specification.

Specify one of the following field names (do not abbreviate):

NODE	Node name
DEVICE	Device name
DIRECTORY	Directory name

NAME	File name
TYPE	File type
VERSION	File version number

parse-type

Specifies the type of parsing to be performed. By default, the F$PARSE function verifies that the directory in the file specification exists on the device in the file specification. However, the existence of the directory is not verified if you provide a **field** argument. Note that the device and directory can be explicitly given in one of the arguments, or can be provided by default.

Also, by default the F$PARSE function translates logical names if they are provided in any of the arguments. The F$PARSE function stops iterative translation when it encounters a logical name with the CONCEALED attribute.

You can change how the F$PARSE function parses a file specification by using one of the following keywords:

NO_CONCEAL	Ignores the "conceal" attribute in the translation of a logical name as part of the file specification; that is, logical name translation does not end when a concealed logical name is encountered.
SYNTAX_ONLY	The syntax of the file specification is checked without verifying that the specified directory exists on the specified device.

DESCRIPTION The F$PARSE function parses file specifications by using the RMS service $PARSE. For more information on the $PARSE service, see the *VMS Record Management Services Manual*.

When you use the F$PARSE function, you can omit those optional arguments to the right of the last argument you specify. However, you must include commas (,) as placeholders if you omit optional arguments to the left of the last argument you specify.

If you omit the device and directory names in the **filespec** argument, the F$PARSE function supplies defaults, first from the **default-spec** argument and second from the **related-spec** argument. If names are not provided by these arguments, the F$PARSE function uses your current default disk and directory.

If you omit the node name, the file name, the file type, or the version number, the F$PARSE function supplies defaults, first from the **default-spec** argument and second from the **related-spec** argument. If names are not provided by these arguments, the F$PARSE function returns a null specification for these fields.

EXAMPLES

1
```
$  SET DEF DISK2:[FIRST]
$  SPEC = F$PARSE("JAMES.MAR","[ROOT]",,,"SYNTAX_ONLY")
$  SHOW SYMBOL SPEC
   SPEC = "DISK2:[ROOT]JAMES.MAR;"
```

> In this example, the F$PARSE function returns the expanded file specification for the file JAMES.MAR. The example uses the SYNTAX_ONLY keyword to request that F$PARSE check the syntax, but should not verify that the [ROOT] directory exists on DISK2.

> The default device and directory are DISK2:[FIRST]. Because the directory name [ROOT] is specified as the **default-spec** argument in the assignment statement, it is used as the directory name in the output string. Note that the default device returned in the output string is DISK2, and the default version number for the file is null. You must place quotation marks (" ") around the arguments JAMES.MAR and ROOT because they are string literals.

> If you had not specified syntax-only parsing, and [ROOT] were not on DISK2, a null string would have been returned.

2
```
$  SET DEFAULT DB1:[VARGO]
$  SPEC = F$PARSE("INFO.COM",,,"DIRECTORY")
$  SHOW SYMBOL SPEC
   SPEC = "[VARGO]"
```

> In this example the F$PARSE function returns the directory name of the file INFO.COM. Note that because the **default-spec** and **related-spec** arguments are omitted from the argument list, commas (,) must be inserted in their place.

3
```
$  SPEC= F$PARSE("DENVER::DB1:[PROD]RUN.DAT",,,"TYPE")
$  SHOW SYMBOL SPEC
   SPEC = ".DAT"
```

> In this example, the F$PARSE function is used to parse a file specification containing a node name. The F$PARSE function returns the file type DAT for the file RUN.DAT at the remote node DENVER.

F$PID

Returns a process identification (PID) number and updates the context symbol to point to the current position in the system's process list.

FORMAT   **F$PID**(context-symbol)

---

return value   A character string containing the PID of a process in the system's list of processes.

ARGUMENT   *context-symbol*
Specifies a symbol that DCL uses to store a pointer into the system's list of processes. The F$PID function uses this pointer to return a PID.

Specify the context symbol by using a symbol. The first time you use the F$PID function in a command procedure, you should use a symbol that is either undefined or equated to the null string (""), or a context symbol that has been created by the F$CONTEXT function.

If the context symbol is undefined or equated to a null string, the F$PID function returns the first PID in the system's process list that it has the privilege to access. That is, if you have GROUP privilege and if the context symbol is null or undefined, the F$PID function returns the PID of the first process in your group. If you have WORLD privilege, the F$PID function returns the PID of the first process in the list. If you have neither GROUP nor WORLD privilege, the F$PID returns the first process that you own. Subsequent calls to F$PID return the rest of the processes on the system you are accessing.

If the context symbol has been created by the F$CONTEXT function, the F$PID function returns the first process name in the system's process list that fits the criteria specified in the F$CONTEXT calls. Subsequent calls to F$PID return only the PIDs of those processes that meet the selection criteria set up by the F$CONTEXT function and that are accessible to your current privileges.

DESCRIPTION   The F$PID function returns a process identification (PID) number and updates the context symbol to point to the current position in the system's process list. You can step through all the processes on a system, or use the lexical function F$CONTEXT to specify selection criteria. The function F$CONTEXT is not required.

The PIDs returned by the F$PID function depend on the privilege of your process. If you have GROUP privilege, the F$PID function returns PIDs of processes in your group. If you have WORLD privilege, the F$PID function returns PIDs of all processes on the system. If you lack GROUP or WORLD privilege, the F$PID function returns only those processes that you own.

The F$CONTEXT function enables the F$PID function to retrieve processes from any node in a VAXcluster.

The first time you use the F$PID function, use a symbol that is either undefined or equated to the null string or to a context symbol that has been created by the F$CONTEXT function. This causes the F$PID function to return the first PID in the system's process list that you have the privilege to access. It also causes the F$PID function to initialize the **context-symbol** argument.

Once the **context-symbol** argument is initialized, each subsequent F$PID returns the next PID in sequence, using the selection criteria set up by the F$CONTEXT function, if any, and updates the context symbol. After the last PID in the process list is returned, the F$PID function returns a null string.

EXAMPLE

```
$ CONTEXT = ""
$ START:
$     PID = F$PID(CONTEXT)
$     IF PID .EQS. "" THEN EXIT
$     SHOW SYMBOL PID
$     GOTO START
```

This command procedure uses the F$PID function to display a list of PIDs. The assignment statement declares the symbol CONTEXT, which is used as the **context-symbol** argument for the F$PID function. Because CONTEXT is equated to a null string, the F$PID function returns the first PID in the process list that it has the privilege to access.

The PIDs displayed by this command procedure depend on the privilege of your process. When run with GROUP privilege, the PIDs of users in your group are displayed. When run with WORLD privilege, the PIDs of all users on the system are displayed. Without GROUP or WORLD privilege, only those processes that you own are displayed.

F$PRIVILEGE

Returns a string value of either TRUE or FALSE, depending on whether your current process privileges match those specified in the argument. You can specify either the positive or negative version of a privilege.

FORMAT   **F$PRIVILEGE**(priv-states)

---

return value   A character string containing the value TRUE or FALSE. The F$PRIVILEGE function returns the string FALSE if any one of the privileges in the **priv-states** argument list is false.

ARGUMENT   *priv-states*
Specifies a character string containing a privilege, or a list of privileges separated by commas (,). For a list of process privileges, see the *VMS DCL Concepts Manual*. Specify any one of the process privileges except [NO]ALL.

DESCRIPTION   Use the F$PRIVILEGE function to identify your current process privileges.

If "NO" precedes the privilege, the privilege must be disabled in order for the function to return a value of TRUE. The F$PRIVILEGE function checks each of the keywords in the specified list, and if the result for any one is false, the string FALSE is returned.

EXAMPLE

```
$  PROCPRIV = F$PRIVILEGE("OPER,GROUP,TMPMBX,NONETMBX")
$  SHOW SYMBOL PROCPRIV
  PROCPRIV = "FALSE"
```

The F$PRIVILEGE function is used to test whether the process has OPER, USER, TMPMBX, and NETMBX privileges.

The process in this example has OPER (operator), GROUP, TMPMBX (temporary mailbox), and NETMBX (network mailbox) privileges. Therefore, a value of FALSE is returned because the process has NETMBX privilege, but NONETMBX was specified in the priv-states list. Although the Boolean result for the other three keywords is true, the entire expression is declared false because the result for NONETMBX was false.

F$PROCESS

Obtains the current process name string. The F$PROCESS function has no arguments, but must be followed by parentheses.

FORMAT

F$PROCESS()

return value

A character string containing the current process name.

ARGUMENTS

None.

EXAMPLE

```
$  NAME = F$PROCESS()
$  SHOW SYMBOL NAME
  NAME = "MARTIN"
```

In this example, the F$PROCESS function returns the current process name and assigns it to the symbol NAME.

F$SEARCH

Searches a directory file and returns the full file specification for a file you specify.

FORMAT   **F$SEARCH**(filespec[,stream-id])

---

return value   A character string containing the expanded file specification for the **filespec** argument. If the F$SEARCH function does not find the file in the directory, the function returns a null string (" ").

ARGUMENTS   *filespec*
Specifies a character string containing the file specification to be searched for. If the device or directory names are omitted, the defaults from your current default disk and directory are used. The F$SEARCH function does not supply defaults for a file name or type. If the version is omitted, the specification for the file with the highest version number is returned. If the **filespec** argument contains wildcards, each time F$SEARCH is called, the next file specification that agrees with the **filespec** argument is returned. A null string is returned after the last file specification that agrees with the **filespec** argument.

stream-id
Specifies a positive integer representing the search stream identification number.

The search stream identification number is used to maintain separate search contexts when you use the F$SEARCH function more than once and when you supply different **filespec** arguments. If you use the F$SEARCH function more than once in a command procedure and if you also use different **filespec** arguments, specify **stream-id** arguments to identify each search separately.

If you omit the **stream-id** argument, the F$SEARCH function assumes an implicit single search stream. That is, the F$SEARCH function starts searching at the beginning of the directory file each time you specify a different **filespec** argument.

DESCRIPTION   The lexical function F$SEARCH invokes the RMS service $SEARCH to search a directory file and return the full file specification for a file you specify. The F$SEARCH function allows you to search for files in a directory by using the RMS service $SEARCH. For more information on the $SEARCH routine, see the *VMS Record Management Services Manual*.

You can use the F$SEARCH function in a loop in a command procedure to return file specifications for all files that match a **filespec** argument containing a wildcard. Each time the F$SEARCH function is executed, it returns the next file specification that matches the file specification that contains a wildcard. After the last file specification is returned, the next

F$SEARCH call returns a null string. When you use the F$SEARCH function in a loop, you must include a wildcard character (* and %) in the **filespec** argument. Otherwise, the F$SEARCH always returns the same file specification.

Note that you must maintain the context of the search stream explicitly (by stating a **stream-id** argument) or implicitly (by omitting the **stream-id** argument and by using the same **filespec** argument each time you execute the F$SEARCH function).

Note: **The lexical function F$SEARCH can return any file that matches the selection criteria you specify, and that exists in the directory at some time between the beginning and the end of the search. Files that are created, renamed, or deleted during the search may or may not be returned.**

EXAMPLES

1
```
$ START:
$    FILE = F$SEARCH("SYS$SYSTEM:*.EXE")
$    IF FILE .EQS. "" THEN EXIT
$    SHOW SYMBOL FILE
$    GOTO START
```

This command procedure displays the file specifications of the latest version of all EXE files in the SYS$SYSTEM directory. (Only the latest version is returned because a wildcard is not used as the version number.) The **filespec** argument SYS$SYSTEM:*.EXE is surrounded by quotation marks (" ") because it is a character string expression.

Because no **stream-id** argument is specified, the F$SEARCH function uses a single search stream. Each subsequent F$SEARCH call uses the same **filespec** argument to return the next file specification of an EXE file from SYS$SYSTEM:. After the latest version of each EXE file has been displayed, the F$SEARCH function returns a null string ("") and the procedure exits.

2
```
$ START:
$    COM = F$SEARCH ("*.COM;*",1)
$    DAT = F$SEARCH ("*.DAT;*",2)
$    SHOW SYMBOL COM
$    SHOW SYMBOL DAT
$    IF (COM.EQS. "") .AND. (DAT.EQS. "") THEN EXIT
$    GOTO START
```

This command procedure searches the default disk and directory for both COM and DAT files. Note that the **stream-id** argument is specified for each F$SEARCH call so that the context for each search is maintained.

The first F$SEARCH call starts searching from the top of the directory file for a file with a type of COM. When it finds a COM file, a pointer is set to maintain the search context. When the F$SEARCH function is used the second time, it again starts searching from the top of the directory file for a file with a type of DAT. When the procedure loops back to the label START, the **stream-id** argument allows F$SEARCH to start searching in

the correct place in the directory file. After all versions of COM and DAT files are returned, the procedure exits.

❸
```
$  FILESPEC = F$SEARCH("TRNTO""SMITH SALLY""::DBA1:[PROD]*.DAT")
$  SHOW SYMBOL FILESPEC
   FILESPEC = "TRNTO"smith password"::DBA1:[PROD]CARS.DAT"
```

This example uses the F$SEARCH function to return a file specification for a file at a remote node. The access control string is enclosed in quotation marks because it is part of a character string expression when it is an argument for the F$SEARCH function. To include quotation marks in a character string expression, you must use two sets of quotation marks.

Note that, when the F$SEARCH function returns a node name containing an access control string, it substitutes the word "password" for the actual user password.

F$SETPRV

Enables or disables specified user privileges. The F$SETPRV function returns a list of keywords indicating user privileges; this list shows the status of the specified privileges before F$SETPRV was executed.

Your process must be authorized to set the specified privilege. For detailed information on privilege restrictions, see the description of the $SETPRV system service in the *VMS System Services Reference Manual.*

FORMAT

F$SETPRV *(priv-states)*

return value

A character string containing keywords for the current process privileges before they were changed by the F$SETPRV function.

ARGUMENT

priv-states

Specifies a character string defining a privilege, or a list of privileges separated by commas (,).

For a list of process privileges, see the *VMS DCL Concepts Manual.*

DESCRIPTION

The lexical function F$SETPRV invokes the $SETPRV system service to enable or disable specified user privileges. The F$SETPRV function returns a list of keywords indicating user privileges; this list shows the status of the specified privileges before F$SETPRV was executed.

The F$SETPRV function returns keywords for your current privileges, whether or not you are authorized to change the privileges listed in the **priv-states** argument. However, the F$SETPRV function enables or disables only the privileges you are authorized to change.

When you run programs or execute procedures that include the F$SETPRV function, be sure that F$SETPRV restores your process to its proper privileged state. For additional information, refer to the examples that follow.

EXAMPLES

1
```
$ OLDPRIV = F$SETPRV("OPER,NOTMPMBX")
$ SHOW SYMBOL OLDPRIV
  OLDPRIV = "NOOPER,TMPMBX"
```

In this example, the process is authorized to change the OPER (operator) and TMPMBX (temporary mailbox) privileges. The F$SETPRV function enables the OPER privilege and disables the TMPMBX privilege. In addition, the F$SETPRV function returns the keywords NOOPER and TMPMBX, showing the state of these privileges before they were changed.

You must place quotation marks (" ") around the list of privilege keywords because it is a string literal.

```
$  SHOW PROCESS/PRIVILEGE

19-APR-1990 15:55:09.60    RTA1:            User: JACKSON

Process privileges:

Process rights identifiers:
 INTERACTIVE
 LOCAL

$  NEWPRIVS = F$SETPRV("ALL, NOOPER")
$  SHOW SYMBOL NEWPRIVS
   NEWPRIVS = "NOCMKRNL,NOCMEXEC,NOSYSNAM,NOGRPNAM,NOALLSPOOL,NODETACH,
       NODIAGNOSE,NOLOG_IO,NOGROUP,NOACNT,NOPRMCEB,NOPRMMBX,NOPSWAPM,
       NOALTPRI,NOSETPRV,NOTMPMBX,NOWORLD,NOMOUNT,NOOPER,NOEXQUOTA,
       NONETMBX,NOVOLPRO,NOPHY_IO,NOBUGCHK,NOPRMGBL,NOSYSGBL,NOPFNMAP,
       NOSHMEM,NOSYSPRV,NOBYPASS,NOSYSLCK,NOSHARE,NOUPGRADE,NODOWNGRADE,
       NOGRPPRV,NOREADALL,NOSECURITY,OPER"
$  SHOW PROCESS/PRIVILEGE

19-APR-1990 15:59:19.30    RTA1:            User: JACKSON

Process privileges:
 CMKRNL               may change mode to kernel
 CMEXEC               may change mode to exec
 SYSNAM               may insert in system logical name table
 GRPNAM               may insert in group logical name table
 ALLSPOOL             may allocate spooled device
 DETACH               may create detached processes
 DIAGNOSE             may diagnose devices
 LOG_IO               may do logical i/o
 GROUP                may affect other processes in same group
 ACNT                 may suppress accounting message
 PRMCEB               may create permanent common event clusters
 PRMMBX               may create permanent mailbox
 PSWAPM               may change process swap mode
 ALTPRI               may set any priority value
 SETPRV               may set any privilege bit
 TMPMBX               may create temporary mailbox
 WORLD                may affect other processes in the world
 MOUNT                may execute mount acp function
 EXQUOTA              may exceed quota
 NETMBX               may create network device
 VOLPRO               may override volume protection
 PHY_IO               may do physical i/o
 BUGCHK               may make bug check log entries
 PRMGBL               may create permanent global sections
 SYSGBL               may create system wide global sections
 PFNMAP               may map to specific physical pages
 SHMEM                may create/delete objects in shared memory
 SYSPRV               may access objects via system protection
 BYPASS               bypasses UIC checking
 SYSLCK               may lock system wide resources
 SHARE                may assign channels to non-shared device
 GRPPRV               group access via system protection
 READALL              may read anything as the owner
 SECURITY             may perform security functions

Process rights identifiers:
 INTERACTIVE
 LOCAL
```

```
$ NEWPRIVS = F$SETPRV(NEWPRIVS)
$ SHOW PROCESS/PRIVILEGE

19-APR-1990 16:05:07.23    RTA1:                 User: JACKSON

Process privileges:
 OPER                        operator privilege

Process rights identifiers:
 INTERACTIVE
 LOCAL
```

In this example, the DCL command SHOW PROCESS/PRIVILEGE is used to determine the current process privileges. Note that the process has no privileges enabled.

The F$SETPRV function is then used to process the ALL keyword and enable all privileges recording the previous state of each privilege in the symbol NEWPRIVS. Next, F$SETPRV processes the NOOPER keyword and disables the OPER (operator) privilege, recording the previous state of OPER in NEWPRIVS. Note that the OPER privilege appears in the returned string twice: first as NOOPER and then as OPER.

Entering the command SHOW PROCESS/PRIVILEGE now shows that the current process has all privileges enabled except OPER.

If the returned string is used as the parameter to F$SETPRV, the process has the OPER privilege enabled. This occurs because the OPER command was present twice in the symbol NEWPRIVS. As a result, F$SETPRV looked at the first keyword NOOPER and disabled the privilege. Finally, after processing several other keywords in the NEWPRIVS string, the OPER keyword is presented, allowing F$SETPRV to enable the OPER privilege.

If you are using the ALL or NOALL keywords to save your current privilege environment, Digital recommends that you perform the following procedure to modify the process for a command procedure:

```
$ CURRENT_PRIVS = F$SETPRV("ALL")
$ TEMP = F$SETPRV("NOOPER")
```

If you use this procedure, you can then specify the following command statement at the end of your command procedure so that the original privilege environment is restored:

```
$ TEMP = F$SETPRV(CURRENT_PRIVS)
```

3
```
$ SAVPRIV = F$SETPRV("NOGROUP")
$ SHOW SYMBOL SAVPRIV
  SAVPRIV = "GROUP"
$ TEST = F$PRIVILEGE("GROUP")
$ SHOW SYMBOL TEST
  TEST = "TRUE"
```

In this example, the process is not authorized to change the GROUP privilege. However, the F$SETPRV function still returns the current setting for the GROUP privilege.

The F$PRIVILEGE function is used to see whether the process has GROUP privilege. The return string, TRUE, indicates that the process has GROUP privilege, even though the F$SETPRV function attempted to disable the privilege.

F$STRING

Returns the string that is equivalent to the specified expression.

FORMAT   **F$STRING***(expression)*

---

return value   A character string equivalent to the specified expression.

ARGUMENT   *expression*
The integer or string expression to be evaluated.

If you specify an integer expression, the F$STRING function evaluates the expression, converts the resulting integer to a string, and returns the result. If you specify a string expression, the F$STRING function evaluates the expression and returns the result.

When converting an integer to a string, the F$STRING function uses decimal representation and omits leading zeros. When converting a negative integer, the F$STRING function places a minus sign at the beginning string representation of the integer.

EXAMPLE

```
$  A = 5
$  B = F$STRING(-2 + A)
$  SHOW SYMBOL B
  B = "3"
```

The F$STRING function in this example converts the result of the integer expression (–2 + A) to the numeric string, "3". First, the F$STRING function evaluates the expression (–2 + A). Note that 5, the value of symbol A, is automatically substituted when the integer expression is evaluated.

After the integer expression is evaluated, the F$STRING function converts the resulting integer, 3, to the string "3". This string is assigned to the symbol B.

F$TIME

Returns the current date and time in absolute time format.

The F$TIME function has no arguments, but must be followed by parentheses.

FORMAT   F$TIME()

return value   A character string containing the current date and time. The returned string has the following fixed, 23-character format:

dd-mmm-yyyy hh:mm:ss.cc

When the current day of the month is any of the values 1 to 9, the first character in the returned string is a blank character. The time portion of the string is always in character position 13, at an offset of 12 characters from the beginning of the string.

Note that you must use the assignment operator (=) to preserve the blank character in the returned string. If you use the string assignment operator (:=), the leading blank is dropped.

ARGUMENTS   *None.*

EXAMPLE

```
$ OPEN/WRITE OUTFILE DATA.DAT
$ TIME_STAMP = F$TIME()
$ WRITE OUTFILE TIME_STAMP
```

This example shows how to use the F$TIME function to time-stamp a file that you create from a command procedure. OUTFILE is the logical name for the file DATA.DAT, which is opened for writing. The F$TIME function returns the current date and time string, and assigns this string to the symbol TIME_STAMP. The WRITE command writes the date and time string to OUTFILE.

F$TRNLNM

Translates a logical name and returns the equivalence name string or the requested attributes of the logical name specified.

FORMAT **F$TRNLNM**(logical-name [,table] [,index] [,mode] [,case] [,item])

---

return value The equivalence name or attribute of the specified logical name. The return value can be a character string or an integer, depending on the arguments you specify with the F$TRNLNM function. If no match is found, a null string ("") is returned.

ARGUMENTS **logical-name**
Specifies a character string containing the logical name to be translated.

table
Specifies a character string containing the logical name table or tables that the F$TRNLNM function should search to translate the logical name. The table argument must be a logical name that translates to a logical name table or to a list of table names.

If you do not specify a table, the default value is LNM$DCL_LOGICAL. That is, the F$TRNLNM function searches the tables whose names are equated to the logical name LNM$DCL_LOGICAL. Unless LNM$DCL_LOGICAL has been redefined for your process, the F$TRNLNM function searches the process, job, group, and system logical name tables, in that order, and returns the equivalence name for the first match found.

index
Specifies the number of the equivalence name to be returned if the logical name has more than one translation. The index refers to the equivalence strings in the order the names were listed when the logical name was defined.

The index begins with zero; that is, the first name in a list of equivalence names is referenced by the index zero.

If you do not specify the **index** argument, the default is zero.

mode
Specifies a character string containing one of the following access modes for the translation: USER (default), SUPERVISOR, EXECUTIVE, or KERNEL.

The F$TRNLNM function starts by searching for a logical name created with the access mode specified in the **mode** argument. If it does not find a match, the F$TRNLNM function searches for the name created with each inner access mode and returns the first match found. For example, two logical names can have the same name, but one name can be created with user access mode and the other name with executive access mode.

If the **mode** argument is USER, the F$TRNLNM function returns the equivalence string for the user-mode, not the executive-mode, logical name.

case

Specifies the type of case translation to be performed. Specify the case argument as either of the following character strings: CASE_BLIND (default) or CASE_SENSITIVE.

If the translation is case blind, the F$TRNLNM function first searches for a logical name with characters of the same case as the **logical-name** argument. If no match is found, the F$TRNLNM function searches for an uppercase version of the **logical-name** argument and the logical names it is searching. The result of the first successful translation is returned.

If the translation is case sensitive, the F$TRNLNM function searches only for a logical name with characters of the same case as the **logical-name** argument. If no exact match is found, the F$TRNLNM function returns a null string ("").

item

Specifies a character string containing the type of information that F$TRNLNM should return about the specified logical name. Specify one of the following items:

Item	Return Type	Information Returned
ACCESS_MODE	String	One of the following access modes associated with the logical name: USER, SUPERVISOR, EXECUTIVE, KERNEL.
CONCEALED	String	TRUE or FALSE to indicate whether the CONCEALED attribute was specified with the /TRANSLATION_ATTRIBUTES qualifier when the logical name was created. The CONCEALED attribute is used to create a concealed logical name.
CONFINE	String	TRUE or FALSE to indicate whether the logical name is confined. If the logical name is confined (TRUE), then the name is not copied to subprocesses. If the logical name is not confined (FALSE), then the name is copied to subprocesses.
CRELOG	String	TRUE or FALSE to indicate whether the logical name was created with the $CRELOG system service or with the $CRELNM system service, using the CRELOG attribute. If the logical name was created with the $CRELOG system service or with the $CRELNM system service, using the CRELOG attribute, then TRUE is returned. Otherwise, FALSE is returned.

Lexical Functions
F$TRNLNM

Item	Return Type	Information Returned
LENGTH	Integer	Length of the equivalence name associated with the specified logical name. If the logical name has more than one equivalence name, the F$TRNLNM function returns the length of the name specified by the index argument.
MAX_INDEX	Integer	The largest index defined for the logical name. The index shows how many equivalence names are associated with a logical name. The index is zero based; that is, the index zero refers to the first name in a list of equivalence names.
NO_ALIAS	String	TRUE or FALSE to indicate whether the logical name has the NO_ALIAS attribute. The NO_ALIAS attribute means that a logical name must be unique within outer access mode.
TABLE	String	TRUE or FALSE to indicate whether the logical name is the name of a logical name table.
TABLE_NAME	String	Name of the table where the logical name was found.
TERMINAL	String	TRUE or FALSE to indicate whether the TERMINAL attribute was specified with the /TRANSLATION_ATTRIBUTES qualifier when the logical name was created. The TERMINAL attribute indicates that the logical name is not a candidate for iterative translation.
VALUE	String	Default. The equivalence name associated with the specified logical name. If the logical name has more than one equivalence name, the F$TRNLNM function returns the name specified by the index argument.

DESCRIPTION

The lexical function F$TRNLNM uses the $TRNLNM system service to translate a logical name and return the equivalence name string, or the requested attributes of the logical name specified. The translation is not iterative; the equivalence string is not checked to determine whether it is a logical name.

When you use the F$TRNLNM function, you can omit optional arguments that can be used to the right of the last argument you specify. However, you must include commas (,) as placeholders if you omit optional arguments to the left of the last argument that you specify.

You can use the F$TRNLNM function in command procedures to save the current equivalence of a logical name and later restore it. You can also use it to test whether logical names have been assigned.

EXAMPLES

1 $ SAVE_DIR = F$TRNLNM("SYS$DISK")+F$DIRECTORY()
  .
  .
  .
     $ SET DEFAULT 'SAVE_DIR'

> The assignment statement concatenates the values returned by the F$DIRECTORY and F$TRNLNM functions, and assigns the resulting string to the symbol SAVE_DIR. The symbol SAVE_DIR consists of a full device and directory name string.
>
> The argument SYS$DISK is enclosed in quotation marks (" ") because it is a character string. (The command interpreter treats all arguments that begin with alphabetic characters as symbols or lexical functions, unless the arguments are enclosed in quotation marks.) None of the optional arguments is specified, so the F$TRNLNM function uses the defaults.
>
> At the end of the command procedure, the original default directory is reset. When you reset the directory, you must place single quotation marks (' ') around the symbol SAVE_DIR to force symbol substitution.

2 $ DEFINE/TABLE=LNM$GROUP TERMINAL 'F$TRNLNM("SYS$OUTPUT")'

> This example shows a line from a command procedure that (1) uses the F$TRNLNM function to determine the name of the current output device and (2) creates a group logical name table entry based on the equivalence string.
>
> You must enclose the argument SYS$OUTPUT in quotation marks because it is a character string.
>
> Also, in this example you must enclose the F$TRNLNM function in single quotation marks to force the lexical function to be evaluated. Otherwise, the DEFINE command does not automatically evaluate the lexical function.

3 $ RESULT = F$TRNLNM("INFILE","LNM$PROCESS",0,"SUPERVISOR",,"NO_ALIAS")
     $ SHOW SYMBOL RESULT
       RESULT = "FALSE"

> In this example, the F$TRNLNM function searches the process logical name table for the logical name INFILE. The function starts the search by looking for the logical name INFILE created in supervisor mode. If no match is found, the function looks for INFILE created in executive mode.
>
> When a match is found, the F$TRNLNM function determines whether the name INFILE was created with the NO_ALIAS attribute. In this case, the NO_ALIAS attribute is not specified.

F$TYPE

Returns the data type of a symbol.

FORMAT

F$TYPE*(symbol-name)*

return value

The string INTEGER is returned if the symbol is equated to an integer, or if the symbol is equated to a string whose characters form a valid integer.

If the symbol has been produced by a call to the F$CONTEXT function with a context type of PROCESS or by a call to the F$PID function, the string returned is PROCESS_CONTEXT. A symbol retains this type until F$CONTEXT is called with the symbol and the CANCEL keyword, or until a null string ("") is returned by a call to F$PID.

Similarly, the return value is the string CLUSTER_SYSTEM_CONTEXT for symbols created by the F$CSID function.

If the symbol is a context symbol, then the return value will be one of the types shown in Table DCL1–13.

Table DCL1–13   Context Symbol Types

Symbol Type	Lexical Creating Symbol
PROCESS_CONTEXT	F$PID or F$CONTEXT (with PROCESS context type)
CLUSTER_SYSTEM_ CONTEXT	F$CSID

The string STRING is returned if the symbol is equated to a character string whose characters do not form a valid integer or whose type is not a context.

If the symbol is undefined, a null string is returned.

ARGUMENT

symbol-name
Specifies the name of the symbol to be evaluated.

EXAMPLES

1
```
$ NUM = "52"
$ TYPE = F$TYPE(NUM)
$ SHOW SYMBOL TYPE
  TYPE = "INTEGER"
```

This example uses the F$TYPE function to determine the data type of the symbol NUM. NUM is equated to the character string "52". Because the characters in the string form a valid integer, the F$TYPE function returns the string INTEGER.

2
```
$  NUM = 52
$  TYPE = F$TYPE(NUM)
$  SHOW SYMBOL TYPE
   TYPE = "INTEGER"
```

> In this example, the symbol NUM is equated to the integer 52. The F$TYPE function shows that the symbol has an integer data type.

3
```
$  CHAR = "FIVE"
$  TYPE = F$TYPE(CHAR)
$  SHOW SYMBOL TYPE
   TYPE = "STRING"
```

> In this example, the symbol CHAR is equated to the character string FIVE. Because the characters in this string do not form a valid integer, the F$TYPE function shows that the symbol has a string value.

4
```
$  x = F$CONTEXT("PROCESS",CTX,"USERNAME","SMITH")
$  TYPE = F$TYPE(CTX)
$  SHOW SYMBOL TYPE
   TYPE = "PROCESS_CONTEXT"
$  x = F$CONTEXT("PROCESS",CTX,"CANCEL")
$  TYPE = F$TYPE(CTX)
$  SHOW SYMBOL TYPE
   TYPE = ""
```

> In this example, the F$TYPE function returns the string PROCESS_ CONTEXT because the symbol has been produced by a call to the F$CONTEXT function with a context type of PROCESS. The symbol returns this type until F$CONTEXT is called with the symbol and the **selection-item** argument value CANCEL.

F$USER

Returns the current user identification code (UIC) in named format as a character string. The F$USER function has no arguments, but must be followed by parentheses.

FORMAT   **F$USER()**

return value   A character string containing the current UIC, including brackets ([ ]). The UIC is returned in the format [group-identifier, member-identifier].

ARGUMENTS   *None.*

EXAMPLE

```
$  UIC = F$USER()
$  SHOW SYMBOL UIC
   UIC = "[GROUP6,JENNIFER]"
```

In this example the F$USER function returns the current user identification code and assigns it to the symbol UIC.

F$VERIFY

Returns an integer value indicating whether the procedure verification setting is currently on or off. If used with arguments, the F$VERIFY function can turn the procedure and image verification settings on or off. You must include the parentheses after the F$VERIFY function whether or not you specify arguments.

FORMAT   **F$VERIFY** *([procedure-value] [,image-value])*

---

return value   The integer 0 if the procedure verification setting is off, or the integer 1 if the procedure verification setting is on.

ARGUMENTS   *procedure-value*
Specifies an integer expression with a value of 1 to turn procedure verification on, or a value of 0 to turn procedure verification off.

When procedure verification is on, each DCL command line in the command procedure is displayed on the output device. Procedure verification allows you to verify that each command is executing correctly.

If you use the **procedure-value** argument, the function first returns the current procedure verification setting. Then the command interpreter turns the procedure verification on or off, as specified by the argument.

image-value
Specifies an integer expression with a value of 1 to turn image verification on, or a value of 0 to turn image verification off.

When image verification is on, data lines in the command procedure are displayed on the output device.

DESCRIPTION   The lexical function F$VERIFY returns an integer value indicating whether the procedure verification setting is currently on or off. If used with arguments, the F$VERIFY function can turn the procedure and image verification settings on or off. You must include the parentheses after the F$VERIFY function whether or not you specify arguments.

Using the F$VERIFY function in command procedures allows you to test the current procedure verification setting. For example, a command procedure can save the current procedure verification setting before changing it and then later restore the setting. In addition, you can construct a procedure that does not display (or print) commands, regardless of what the initial state of verification is.

When you use the F$VERIFY function, you can specify zero, one, or two arguments. If you do not specify any arguments, neither of the verification settings is changed. If you specify only the **procedure-value** argument,

both procedure and image verification are turned on (if the value is 1) or off (if the value is 0).

If you specify both arguments, procedure and image verification are turned on or off independently. If you specify the **image-value** argument alone, only image verification is turned on or off. If you specify the **image-value** argument alone, you must precede the argument with a comma (,).

You can also use the F$ENVIRONMENT function with VERIFY_ PROCEDURE or VERIFY_IMAGE as the argument. With the F$ENVIRONMENT function, you can determine either the procedure or image verification setting; the F$VERIFY function determines only the procedure verification setting.

DCL performs the F$VERIFY function even if it appears after a comment character, if it is enclosed in single quotation marks (' '). This is the only processing that DCL performs within a comment.

EXAMPLES

1
```
$ SAVE_PROC_VERIFY = F$ENVIRONMENT("VERIFY_PROCEDURE")
$ SAVE_IMAGE_VERIFY = F$ENVIRONMENT("VERIFY_IMAGE")
$ SET NOVERIFY
     .
     .
     .
$ TEMP = F$VERIFY(SAVE_PROC_VERIFY, SAVE_IMAGE_VERIFY)
```

This example shows an excerpt from a command procedure. The first assignment statement assigns the current procedure verification setting to the symbol SAVE_PROC_VERIFY. The second assignment statement assigns the current image verification setting to the symbol SAVE_IMAGE_VERIFY.

Then, the SET NOVERIFY command disables procedure and image verification. Later, the F$VERIFY function resets the verification settings, using the original values (equated to the symbols SAVE_PROC_VERIFY and SAVE_IMAGE_VERIFY). The symbol TEMP contains the procedure verification before it is changed with the F$VERIFY function. (In this example the value of TEMP is not used.)

2
```
$ VERIFY = F$VERIFY(0)
     .
     .
     .
$ IF VERIFY .EQ. 1 THEN SET VERIFY
```

This example shows an excerpt from a command procedure that uses the F$VERIFY function to save the current procedure verification setting and to turn both procedure and image verification off. At the end of the command procedure, if procedure verification was originally on, both the procedure and image verification are turned on.

LIBRARY

Invokes the Librarian Utility, which creates, modifies, or describes an object, macro, help, text, or shareable image library. For a complete description of the Librarian Utility, see the *VMS Librarian Utility Manual*.

FORMAT   **LIBRARY**   *library-filespec [input-filespec[,...]]*

LICENSE

Invokes the License Management Utility, which manages software licenses on the VMS operating system. For a complete description of the License Management Utility, see the *VMS License Management Utility Manual*.

FORMAT   **LICENSE** *subcommand parameter*

LINK

Invokes the VMS Linker, which links one or more object modules into a program image and defines execution characteristics of the image. For a complete description of the linker, including more information about the LINK command, see the *VMS Linker Utility Manual.*

FORMAT

LINK *filespec[,...]*

PARAMETER

filespec[,...]

Specifies one or more input files (wildcard characters not allowed). Input files can be object modules, libraries to be searched for external references or from which specific modules are to be included, shareable images to be included in the output image, or option files to be read by the linker. If you specify more than one input file, separate the file specifications with either commas (,) or plus signs (+). In either case, the linker creates a single image file.

If you omit the file type in an input file specification, the linker supplies default file types, based on the nature of the file. For object modules, the file type OBJ is assumed.

DESCRIPTION

Before a source-language program can run on the VMS operating system, it must be translated into object code and then linked. The VMS Linker binds the object modules, together with any other necessary information, into an executable image.

To invoke the VMS Linker from DCL level, enter the LINK command and the command parameter.

For an executable image, the command parameter specifies one or more input files including object modules to be linked, libraries to be searched for external references or from which specific modules are to be included, and option files to be read by the linker. Note that you cannot specify a shareable image input file from the command line; it can only be specified from a statement within an options file that you name on the command line.

If you name several input files on the command line, separate the file specifications with commas or plus signs.

For a shareable image, the command parameter specifies the name of the shareable image being created.

You can direct output to different types of files by using appropriate qualifiers, such as /EXECUTABLE, /SHAREABLE, /MAP, and /SYMBOL_TABLE. By default, linker output and messages are directed to the SYS$OUTPUT device. Unless you specify otherwise, output files take the name of the first input file in the command parameter.

The command line can include qualifiers that either modify the command itself, or modify a particular file in a list.

A command qualifier modifies the command itself and can be located anywhere on the command line. Its position does not alter its function.

Positional qualifiers indicate which of the files in the command parameter list are to be the object of the specified action. If you position the qualifier next to the command, all listed files are affected. To affect one or more files selectively, position the qualifier immediately after the appropriate file specifications.

If you specify incompatible qualifiers, the linker either ignores the command and displays an error message or it may ignore the incompatibility and permit the linking operation to continue.

QUALIFIERS

/BRIEF

Requests the linker to produce a brief map (memory allocation) file; the /BRIEF qualifier is valid only with the /MAP qualifier.

A brief form of the map contains the following information:

- A summary of the image characteristics

- A list of all object modules included in the image

- A summary of link-time performance statistics

/CONTIGUOUS
/NOCONTIGUOUS (default)

Controls whether the output image file is contiguous.

/CROSS_REFERENCE
/NOCROSS_REFERENCE (default)

Controls whether the memory allocation listing (map) contains a symbol cross-reference list with entries for each global symbol referenced in the image, its value, and all modules in the image that refer to it.

/DEBUG[=filespec]
/NODEBUG (default)

Controls whether a debugger is included in the output image.

If the object module contains local symbol table or traceback information, you can specify the /DEBUG qualifier to include the information in the image as well. If the object module does not contain local symbol table or traceback information, only global symbols are available for symbolic debugging.

If you specify the /DEBUG qualifier, the VAX Symbolic Debugger is linked with the image by default. However, you can use the *filespec* parameter to specify an alternate debugger (wildcard characters are not allowed).

For information on using the VMS Debugger, see the *VMS Debugger Manual*.

/EXECUTABLE[=filespec]
/NOEXECUTABLE

Controls whether the linker creates an executable image.

By default the linker creates an executable image with the same file name as the first input file and a file type of EXE, but this qualifier gives you the option of assigning the image a file specification. Wildcard characters are not allowed.

The placement of the command qualifier determines the output file specification defaults.

You can use the /NOEXECUTABLE or the /EXECUTABLE=NL: qualifier to test a set of qualifiers, options, or input object modules without creating an image file. However, it is recommended that you use the /EXECUTABLE=NL: qualifier, because the linker will not process certain other qualifiers if the /NOEXECUTABLE qualifier is used.

/FULL

Produces a full memory allocation (map) listing; the /FULL qualifier is valid only with the /MAP qualifier.

A full listing contains the following information:

* All the information included in a brief listing (see the description of the /BRIEF qualifier)

* Detailed descriptions of each program section and image section in the image file

* Lists of global symbols by name and by value

/HEADER

Provides a system image header when used with the /SYSTEM qualifier. All other images always have headers. However, by default, system images do not have headers.

/INCLUDE=(module-name[,...])
Positional qualifier.

Selects modules from the associated object module library or image library as input to the linking operation. No wildcard characters are allowed in the module name specifications.

At least one module name must be specified. If you specify only one module, you can omit the parentheses.

If you specify the /INCLUDE qualifier, you can also specify the /LIBRARY qualifier; the library is then searched for unresolved references.

/LIBRARY
Positional qualifier.

Indicates that the associated input file is a library (default file type OLB) whose modules should be searched to resolve undefined symbols. You are not permitted to specify a library as the first input file unless you also specify the /INCLUDE qualifier to indicate which modules in the library are to be included in the input. If you use both the /INCLUDE and the /LIBRARY qualifiers, the explicit inclusion of modules occurs first, then the library is used to search for unresolved references.

/MAP[=filespec]
/NOMAP

Controls whether a memory allocation listing (map) is produced and gives you the option of assigning it a file specification. In interactive mode, the default qualifier is /NOMAP; in batch mode, the default qualifier is /MAP.

You can specify the map's contents using either the /BRIEF, the /FULL, or the /CROSS_REFERENCE qualifier. If you do not specify any of these qualifiers, the map contains the following information:

- All the information contained in a brief listing (see /BRIEF)

- A list of user-defined global symbols by name

- A list of user-defined program sections

When you specify the /MAP qualifier, you can control the defaults applied to the output file specification, as described in the *VMS DCL Concepts Manual*.

/OPTIONS
Positional qualifier.

Indicates that the associated input file (default file type OPT) contains a list of linking options.

For complete details on the contents of an options file, see the *VMS Linker Utility Manual*.

/P0IMAGE

Creates an image that is stored only in P0 address space together with the stack and the VMS RMS buffers that usually go in P1 address space. The /P0IMAGE qualifier is used to create executable images that modify P1 address space. For a description of P0 and P1 address space, see the *VAX Architecture Handbook*.

/PROTECT

Creates a protected shareable image that can execute privileged change-mode instructions even when it is linked to a nonprivileged executable image. The /PROTECT qualifier must be used with the /SHAREABLE qualifier.

/SELECTIVE_SEARCH
Positional qualifier.

Omits from the output image symbol table all symbols from the associated input object module that are not needed to resolve outstanding references. These symbols are also excluded from the symbol table file, if the /SYMBOL_TABLE qualifier is specified. The binary code in the object module is always included.

/SHAREABLE[=filespec]
/NOSHAREABLE
Command qualifier.

Creates a shareable image file. By default, the linker creates an executable image. Optionally, you can designate a name for the output file; however, wildcard characters are not permitted.

Shareable images are not executable; however, they can be linked with object modules to create executable images. If you specify both the /EXECUTABLE and /SHAREABLE qualifiers, the /SHAREABLE qualifier takes precedence.

When you specify the /SHAREABLE qualifier, you can control the defaults applied to the output file specification by the placement of the qualifier in the command.

To specify an input shareable image, the /SHAREABLE qualifier must be used as an input file qualifier in an options file. See the description of the linker in the *VMS Linker Utility Manual*.

/SHAREABLE
/SHAREABLE=NOCOPY
Positional qualifier. Use this positional qualifier only within an options file.

Identifies an input file as a shareable image file. The keyword NOCOPY tells the linker not to bind a private copy of the shareable image to the executable image. The /SHAREABLE and /SHAREABLE=NOCOPY qualifiers are equivalent.

/SYMBOL_TABLE[=filespec]
/NOSYMBOL_TABLE (default)
Controls whether a symbol table object module file (default file type STB) is created that contains symbol definitions for all global symbols in the image being linked. The symbol table file can be subsequently specified in LINK commands to provide the symbol definitions to other images.

If you specify the /DEBUG qualifier, the linker creates a separate symbol table file and it includes within the image the global symbol definitions that are used by the debugger.

When you specify the /SYMBOL_TABLE qualifier, you can control the defaults applied to the output file specification. Optionally, you can designate a name for the symbol table file, but you cannot use wildcard characters.

/SYSLIB (default)
/NOSYSLIB
Controls whether the default system libraries, SYS$LIBRARY:IMAGELIB.OLB and then SYS$LIBRARY:STARLET.OLB, are automatically searched for unresolved references in the input files.

/SYSSHR (default)
/NOSYSSHR
Controls whether the default system shareable image library, SYS$LIBRARY:IMAGELIB.OLB, is automatically searched for unresolved references in the input files. By default, the linker automatically searches the object module library SYS$LIBRARY:STARLET.OLB and then SYS$LIBRARY:IMAGELIB.OLB when it cannot resolve references in the input files.

/SYSTEM[=base-address]
/NOSYSTEM (default)

Controls whether a system image is produced. The /SYSTEM qualifier produces a system image and optionally assigns it a base address. You cannot use the /SYSTEM qualifier with either the /SHAREABLE qualifier or the /DEBUG qualifier. A system image cannot be run with the RUN command; it must be bootstrapped or otherwise loaded into memory.

The base address specifies where the image is to be loaded in virtual memory. It can be expressed in decimal, hexadecimal, or octal format, using the radix specifiers %D, %X, or %O, respectively. The default base address is %X80000000.

System images are intended for special purposes, such as standalone operating system diagnostics. When the linker creates a system image, it orders the program sections in alphanumeric order and ignores all program section attributes.

/TRACEBACK (default)
/NOTRACEBACK

Controls whether traceback information is included in the image file to help the system trace the call stack when an error occurs.

If you specify the /DEBUG qualifier, the /TRACEBACK qualifier is assumed.

/USERLIBRARY[=(table[,...])]
/USERLIBRARY=ALL (default)
/NOUSERLIBRARY

Specifies which user-defined default libraries (process, group, system or, by default, all three) the linker searches after it has searched any specified user libraries. (The discussion of the linker in the *VMS Linker Utility Manual* explains user-defined default libraries.) You can specify the following tables for the linker to search:

ALL	By default, the linker searches the process, group, and system logical name tables for user-defined library definitions.
GROUP	The linker searches the group logical name table for user-defined library definitions.
NONE	The linker does not search any logical name table; this specification is equivalent to using the /NOUSERLIBRARY qualifier.
PROCESS	The linker searches the process logical name table for user-defined library definitions.
SYSTEM	The linker searches the system logical name table for user-defined library definitions.

The /NOUSERLIBRARY qualifier tells the linker not to search any user-defined default libraries.

EXAMPLES

1 $ LINK ORION

> The LINK command in this example links the object module in the file
> ORION.OBJ and creates an executable image named ORION.EXE.

2 $ LINK/MAP/FULL DRACO,CYGNUS,LYRA

> The LINK command in this example links the modules DRACO.OBJ,
> CYGNUS.OBJ, and LYRA.OBJ and creates an executable image named
> DRACO.EXE. The /MAP and /FULL qualifiers request a full map of the
> image, with descriptions of each program section, lists of global symbols by
> name and by value, and a summary of the image characteristics. The map
> file is named DRACO.MAP.

3 $ LINK [SSTEST]SERVICE/INCLUDE=DRACO, -
   _$ []CYGNUS/EXECUTABLE

> The LINK command in this example links the object module DRACO
> from the library SERVICE.OLB in the directory SSTEST with the module
> CYGNUS.OBJ in the current default directory. The executable image is
> named CYGNUS.EXE. The placement of the /EXECUTABLE qualifier
> provides the output file name default.

4 $ LINK/MAP/CROSS_REFERENCE/EXECUTABLE=DBGWEATH -
   _$ /DEBUG -
   _$ WEATHER,MATHLIB/LIBRARY
    $ RUN DBGWEATH

                    VAX DEBUG V5.4

    %DEBUG-I-INITIAL, language is FORTRAN, module set to 'WEATHER'
    DBG>

> The LINK command in this example links the object module
> WEATHER.OBJ with the debugger. If any unresolved references are
> encountered, the linker searches the library MATHLIB.OLB before
> searching the system library. The /CROSS_REFERENCE qualifier
> requests a cross-reference listing in the map file; the map file is named,
> by default, WEATHER.MAP. The /EXECUTABLE qualifier requests the
> linker to name the output file DBGWEATH.EXE. The RUN command
> executes the image; the message from the debugger indicates that it is
> ready to accept debug commands.

LOGIN Procedure

Initiates an interactive terminal session.

FORMAT

Ctrl/C

Ctrl/Y

Return

DESCRIPTION
There is no LOGIN command. You signal your intention to access the system by pressing Ctrl/C, Ctrl/Y, or the Return key on a terminal not currently in use. The system prompts for your user name and your password (and your secondary password, if you have one) and then validates them.

Specify the optional qualifiers immediately after you type your user name; then press the Return key to get the password prompts.

The login procedure performs the following functions:

- Validates your right to access the system by checking your user name and passwords against the entries in the system's user authorization file (UAF).

- Establishes the default characteristics of your terminal session based on your user name entry in the UAF.

- Executes the command procedure file SYS$SYLOGIN.COM if one exists.

- Executes either the command procedure file named LOGIN.COM if one exists in your default directory, or the command file defined in the UAF, if any.

Some systems are set up with a retry facility for users who are accessing the system from remote or dialup locations. With these systems, when you make a mistake typing your user name or password, the system allows you to reenter the information. To reenter your login information, press the Return key. The system displays the user name prompt again. Now retype your user name and press the Return key to send the information to the system. The system displays the password prompt. (There is both a limit to the number of times you can retry to enter your login information and a time limit between tries.)

QUALIFIERS
/CLI=command-language-interpreter
Specifies the name of an alternate command language interpreter (CLI) to override the default CLI listed in the UAF. The CLI you specify must be located in SYS$SYSTEM and have the file type EXE.

If you do not specify a command interpreter by using the /CLI qualifier and you do not have a default CLI listed in the UAF, the system supplies the qualifier /CLI=DCL by default.

/COMMAND[=filespec] (default)
/NOCOMMAND

Controls whether to execute your default login command procedure when you log in. Use the /COMMAND qualifier to specify the name of an alternate login command procedure. If you specify a file name without a file type, the default file type COM is used. If you specify the /COMMAND qualifier and omit the file specification, your default login command procedure is executed.

Use the /NOCOMMAND qualifier if you do not want your default login command procedure to be executed.

/DISK=device-name[:]

Specifies the name of a disk device to be associated with the logical device SYS$DISK for the terminal session. This specification overrides the default SYS$DISK device established in the UAF.

/NEW_PASSWORD

Requires that you change the account password before logging in (as if the password had expired). Use this qualifier as a shortcut if you had intended to change your password after login, or if you suspect that your password has been detected.

/TABLES=(command-table[,...])
/TABLES=DCLTABLES (default)

Specifies the name of an alternate CLI table to override the default listed in the UAF. This table name is considered a file specification. The default device and directory is SYS$SHARE and the default file type is EXE.

If a logical name is used, the table name specification must be defined in the system logical name table.

If the /CLI qualifier is set to DCL or MCR, the /TABLES qualifier defaults to the correct value. If the /TABLES qualifier is specified without the /CLI qualifier, the CLI specified in the user's UAF will be used.

EXAMPLES

1
```
     Ctrl/Y
Username:   SMITHSON
Password:   <PASSWORD>
```

In this example, pressing Ctrl/Y allows you to access the operating system, which immediately prompts for a user name. After validating the user name, the system prompts for the password but does not echo it.

LOGIN Procedure

2   `Return`
```
Username:  HIGGINS/DISK=USER$
Password:  <PASSWORD>
       Welcome to VAX/VMS Version 5.4 on node JUPITER
       Last interactive login on Tuesday, 24-APR-1990 09:16:47.08
       Last non-interactive login on Monday, 23-APR-1990 17:32:34.27
$  SHOW DEFAULT
  USER$:[HIGGINS]
```

> In this example, the /DISK qualifier requests that the default disk for the terminal session be DISK2. The SHOW DEFAULT command shows that USER$ is the default disk.

3   `Ctrl/C`
```
Username:  LIZA/CLI=MCR/COMMAND=ALTLOGIN.COM
Password:  <PASSWORD>
       Welcome to VAX/VMS Version 5.4 on node JUPITER
       Last interactive login on Tuesday, 24-APR-1990 09:16:47.08
       Last non-interactive login on Monday, 23-APR-1990 17:32:34.27
>
```

> In this example, the /CLI qualifier requests the alternate MCR command interpreter. The /COMMAND qualifier indicates that the login command file ALTLOGIN.COM is to be executed instead of the default login command file.
>
> The right angle-bracket prompt (>) indicates that MCR is active and expects an MCR command.

4   `Return`
```
Username:  XENAKIS
Password:  <PASSWORD>
Password:  <PASSWORD>
       Welcome to VAX/VMS Version 5.4 on node JUPITER
       Last interactive login on Tuesday, 24-APR-1990 09:16:47.08
       Last non-interactive login on Monday, 23-APR-1990 17:32:34.27
$
```

> In this example, the second password prompt indicates that the user has a secondary password, which must be entered to access the system.

5   `Return`
```
Username:  JONES
Password:  <PASSWORD>
User authorization failure
```
`Return`
```
Username:  JONES
Password:  <PASSWORD>
       Welcome to VAX/VMS Version 5.4 on node JUPITER
       Last interactive login on Tuesday, 24-APR-1990 09:16:47.08
       Last non-interactive login on Monday, 23-APR-1990 17:32:34.27
       1 failure since last successful login.
$
```

> This example shows the "User authorization failure" message, which indicates that the password has been entered incorrectly. After you successfully log in, a message is displayed showing the number of login failures since your last successful login. This message is displayed only if login failures have occurred.

6 `Return`
```
Username:  JOYCE
Password:  <PASSWORD>
        Welcome to VAX/VMS Version 5.4 on node JUPITER
        Last interactive login on Tuesday, 24-APR-1990 09:16:47.08
        Last non-interactive login on Monday, 23-APR-1990 17:32:34.27
        WARNING - Primary password has expired; update immediately.
$
```

> This example shows the WARNING message, which indicates that your
> primary password has expired. You must use the SET PASSWORD
> command to change your password before logging out, or you will be
> unable to log in again.
>
> For more information on changing your password, see the description of
> the SET PASSWORD command in this manual.

7 `Return`
```
Username:  MIHALY/NEW_PASSWORD
Password:  <PASSWORD>
Password:  <PASSWORD>
        Welcome to VAX/VMS Version V5.4 on node JUPITER
        Last interactive login on Tuesday, 24-APR-1990 09:16:47.08
        Last non-interactive login on Monday, 23-APR-1990 17:32:34.27
    Your password has expired; you must set a new password to log in.

        Old password:  <PASSWORD>

        New password:  <PASSWORD>

        Verification:  <PASSWORD>
```

> In this example, the user enters the /NEW_PASSWORD qualifer after
> the user name MIHALY. The system then forces the user to set a new
> password immediately after login. The prompts are the same as those
> provided when you enter the DCL command SET PASSWORD from the
> command line.

LOGOUT

Terminates an interactive terminal session.

FORMAT	**LOGOUT**

DESCRIPTION You must use the LOGOUT command to end a terminal session. Under most circumstances, if you turn the power off at your terminal or hang up your telephone connection without using the LOGOUT command, you remain logged in.

When you use the SET HOST command to log in to a remote processor, you generally need to use the LOGOUT command to end the remote session.

QUALIFIERS */BRIEF*
Prints a brief logout message (process name, date, and time) or a full logout message (a brief message plus accounting statistics).

/FULL
Requests the long form of the logout message. When you specify the /FULL qualifier, the command interpreter displays a summary of accounting information for the terminal session. The default qualifier for a batch job is /FULL.

/HANGUP
/NOHANGUP
Determines, for dialup terminals, whether the phone hangs up whenever you log out. By default, the setting of the /HANGUP qualifier for your terminal port determines whether the line is disconnected. Your system manager determines whether you are permitted to use this qualifier.

EXAMPLES

1 ```
$ LOGOUT
  HIGGINS   logged out at 19-APR-1990 17:48:56.73
```

In this example, the LOGOUT command uses the default brief message form. No accounting information is displayed.

```
❷    $  LOGOUT/FULL
        HIGGINS     logged out at 19-APR-1990 14:23:45.30
     Accounting information:
     Buffered I/O count:         22    Peak working set size:      90
     Direct I/O count:           10    Peak virtual size:          69
     Page faults:                68    Mounted volumes:             0
     Charged CPU time: 0 00:01:30.50  Elapsed time:      0 04:59:02.63
     Charged vector CPU time: 0 00:00:21.62
```

In this example, the LOGOUT command with the /FULL qualifier displays
a summary of accounting statistics for the terminal session.

MACRO

Invokes the VAX MACRO assembler to assemble one or more assembly language source files.

FORMAT   **MACRO** *filespec[,...]*

PARAMETER   *filespec[,...]*
Specifies a VAX MACRO assembly language source file to be assembled. If you specify more than one file, separate the file specifications with either commas (,) or plus signs (+). File specifications separated by commas cause the MACRO assembler to produce an object file (and, if indicated, a listing file) for each specified file. File specifications separated by plus signs are concatenated into one input file and produce a single object file (and listing file).

You cannot include a wildcard character in a file specification. For each file specification, the MACRO command supplies a default file type of MAR. The MACRO assembler creates output files of one version higher than the highest version existing in the target directory.

DESCRIPTION   The MACRO command invokes the VAX MACRO assembler to assemble one or more assembly language source files.

The qualifiers to the MACRO command serve as either command (global) qualifiers or positional qualifiers. A **command qualifier** affects all the files specified in the MACRO command. A **positional qualifier** affects only the file that it qualifies. All MACRO qualifiers except the /LIBRARY and /UPDATE qualifiers are usable as either command or positional qualifiers. The /LIBRARY and /UPDATE qualifiers are positional qualifiers only.

See the qualifier descriptions for restrictions.

For a complete functional description of the VAX MACRO assembler directives, see the *VAX MACRO and Instruction Set Reference Manual*.

QUALIFIERS   */ANALYSIS_DATA[=filespec]*
/NOANALYSIS_DATA (default)
Controls whether the assembler creates an analysis data file for the VAX Source Code Analyzer (SCA), and optionally provides the file specification.

By default, the assembler does not create an analysis data file. If you specify the /ANALYSIS_DATA qualifier without a file specification, the assembler creates a file with the same file name as the first input file for the MACRO command. The default file type for analysis data files is ANA. When you specify the /ANALYSIS_DATA qualifier, you can control the defaults applied to the output file specification by the placement of the qualifier in the command line.

/CROSS_REFERENCE[=(function[,...])]
/NOCROSS_REFERENCE (default)

Controls whether a listing is produced of the locations in the source file where the specified function (or functions) is defined or referenced. If you specify only one function, you can omit the parentheses.

You can specify the following functions:

ALL	Cross-references directives, macros, operation codes, registers, and symbols
DIRECTIVES	Cross-references directives
MACROS	Cross-references macros
OPCODES	Cross-references operation codes
REGISTERS	Cross-references registers
SYMBOLS	Cross-references symbols

Because the assembler writes the cross-references to the listing file, you must specify the /LIST qualifier with the /CROSS_REFERENCE qualifier. If you specify no functions in the /CROSS_REFERENCE qualifier, the assembler assumes the default value of /CROSS_REFERENCE=(MACROS,SYMBOLS). The /NOCROSS_REFERENCE qualifier excludes the cross-reference listing.

/DEBUG[=option]
/NODEBUG (default)

Includes or excludes local symbols in the symbol table or traceback information in the object module. You can replace the /ENABLE and /DISABLE qualifiers with the /DEBUG and /NODEBUG qualifiers when you use the appropriate DEBUG and TRACEBACK options. The /DEBUG or the /NODEBUG qualifier overrides debugging characteristics set with the .ENABLE or .DISABLE assembler directives.

You can specify one or more of the following options:

ALL	Includes in the object module all local symbols in the symbol table, and provides all traceback information for the debugger. This option is equivalent to /ENABLE=(DEBUG,TRACEBACK).
NONE	Makes local symbols and traceback information in the object module unavailable to the debugger. This option is equivalent to /DISABLE=(DEBUG,TRACEBACK).
SYMBOLS	Makes all local symbols in the object module available to the debugger. Makes traceback information unavailable to the debugger. This option is equivalent to /ENABLE=DEBUG and /DISABLE=TRACEBACK together.
TRACEBACK	Makes traceback information in the object module available to the debugger and local symbols unavailable to the debugger. This option is equivalent to /ENABLE=TRACEBACK and /DISABLE=DEBUG together.

If you specify no options to the /DEBUG qualifier, it assumes the default value of /DEBUG=ALL.

/DIAGNOSTICS[=filespec]
/NODIAGNOSTICS (default)

Creates a file containing assembler messages and diagnostic information. If you omit the file specification, the default file name is the same as the source program; the default file type is DIA.

No wildcard characters are allowed in the file specification.

The diagnostics file is reserved for use with Digital layered products, such as the VAX Language-Sensitive Editor (LSE).

/DISABLE=(function[,...])
/NODISABLE

Provides initial settings for the functions disabled by the .DISABLE assembler directive. You can specify one or more of the following functions:

ABSOLUTE	Assembles relative addresses as absolute addresses.
DEBUG	Includes local symbol table information in the object file for use with the debugger.
GLOBAL	Assumes undefined symbols to be external symbols.
SUPPRESSION	Suppresses listing of unreferenced symbols in the symbol table.
TRACEBACK	Provides traceback information to the debugger.
TRUNCATION	Truncates floating-point numbers (if truncation is disabled, numbers are rounded).
VECTOR	Enables the assembler to accept and correctly process vector code.

If you specify only one function, you can omit the parentheses. If you specify no functions in the /DISABLE qualifier, it assumes the default value of /DISABLE=(ABSOLUTE,DEBUG, TRUNCATION, VECTOR). The /NODISABLE qualifier has the same effect as not specifying the /DISABLE qualifier, or negates the effects of any /DISABLE qualifiers specified earlier in the command line.

/ENABLE=(function[,...])
/NOENABLE

Provides initial settings for the functions controlled by the .ENABLE assembler directive.

The /NOENABLE qualifier has the same effect as not specifying the /ENABLE qualifier, or negates the effects of any /ENABLE qualifiers specified earlier in the command line. You can specify one or more of the functions listed in the description of the /DISABLE qualifier. If you specify only one function, you can omit the parentheses. If you specify no functions in the /DISABLE qualifier, it assumes the default value of /ENABLE=(GLOBAL,TRACEBACK,SUPPRESSION).

/LIBRARY
/NOLIBRARY
Positional qualifier. The /LIBRARY qualifier cannot be used with the /UPDATE qualifier.

The associated input file to the /LIBRARY qualifier must be a macro library. The default file type is MLB. The /NOLIBRARY qualifier has the same effect as not specifying the /LIBRARY qualifier, or negates the effects of any /LIBRARY qualifiers specified earlier in the command line.

The assembler can search up to 16 libraries, one of which is always STARLET.MLB. This number applies to a particular assembly, not necessarily to a particular MACRO command. If you enter the MACRO command so that more than one source file is assembled, but the source files are assembled *separately*, you can specify up to 16 macro libraries for each separate assembly. More than one macro library in an assembly causes the libraries to be searched in reverse order of their specification.

A macro call in a source program causes the assembler to begin the following sequence of searches:

1 An initial search of the libraries specified with the .LIBRARY directive. The assembler searches these libraries in the reverse order of that in which they were declared.

2 If the macro definition is not found in any of the libraries specified with the .LIBRARY directive, a search of the libraries specified in the MACRO command line (in the reverse order in which they were specified).

3 If the macro definition is not found in any of the libraries specified in the command line, a search of STARLET.MLB.

/LIST[=filespec]
/NOLIST
Creates or omits an output listing, and optionally provides an output file specification for it. The default file type for the listing file is LIS. No wildcard characters are allowed in the file specification.

An interactive MACRO command does not produce a listing file by default. The /NOLIST qualifier, present either explicitly or by default, causes errors to be reported on the current output device.

The /LIST qualifier is the default for a MACRO command in a batch job. The /LIST qualifier allows you to control the defaults applied to the output file specification by the placement of the qualifier in the command line. For more information on entering output file qualifiers, see the *VMS DCL Concepts Manual*.

/OBJECT[=filespec]
/NOOBJECT
Creates or omits an object module. It also defines the file specification. By default, the assembler creates an object module with the same file name as the first input file. The default file type for object files is OBJ. No wildcard characters are allowed in the file specification.

The /OBJECT qualifier controls the defaults applied to the output file specification by the placement of the qualifier in the command line. For more information on entering output file qualifiers, see the *VMS DCL Concepts Manual*.

/SHOW[=(function[,...])]
/NOSHOW[=(function[,...])]

Provides initial settings for the functions controlled by the assembler directives .SHOW and .NOSHOW.

You can specify one or more of the following functions:

CONDITIONALS   Lists unsatisfied conditional code associated with .IF and .ENDC MACRO directives.

CALLS   Lists macro calls and repeat range expansions.

DEFINITIONS   Lists macro definitions.

EXPANSIONS   Lists macro expansions.

BINARY   Lists binary code generated by the expansion of macro calls.

If you specify more than one function, separate each with a comma and enclose the list in parentheses. If you specify no functions in the /SHOW qualifier, it increments the listing level count; the /NOSHOW qualifier decrements the count in similar circumstances. Because these qualifiers contribute to the listing file, you must also specify the /LIST qualifier when you use them. If you do not specify the /SHOW qualifier, the MACRO command assumes a default of /SHOW=(CONDITIONALS,CALLS,DEFINITIONS). If you specify only one function, you can omit the parentheses.

/UPDATE[=(update-filespec[,...])]
/NOUPDATE

Positional qualifier. The /UPDATE qualifier cannot be used with the /LIBRARY qualifier.

Updates the input file it qualifies by using the SUMSLP batch editor and the specified update file or files. By default, the assembler assumes that the update file has the same file name as the input source file and a file type of UPD. You cannot include a wildcard character in the file specifications. If it cannot find a specified update file, the assembler prints an informational message and continues the assembly.

If you specify only one update file, you can omit the parentheses. If you specify more than one update file, the assembler merges the contents into a single list of updates before applying the updates to the source file.

The /NOUPDATE qualifier has the same effect as not specifying the /UPDATE qualifier, or negates any /UPDATE qualifiers specified earlier in the command line. The input source file and update files are not changed by the update operation. The effects of the update appear in the compiled output. If you specify the /LIST qualifier with the /UPDATE qualifier, the assembler writes an audit trail of the changes to the listing file.

EXAMPLES

1 $ MACRO/LIST CYGNUS, LYRA/OBJECT=LYRAN + MYLIB/LIBRARY

> In this example, the MACRO command requests two separate assemblies. Using MAR as the default file type, MACRO assembles CYGNUS.MAR to produce CYGNUS.LIS and CYGNUS.OBJ. Then it assembles LYRA.MAR and creates a listing file named LYRA.LIS and an object module named LYRAN.OBJ. The default output file type for a listing is LIS.
>
> The command requests the search of the MYLIB library file in the current directory for macro definitions.

2 $ MACRO ORION

> MACRO assembles the file ORION.MAR and creates an object file named ORION.OBJ. Executing the command in a batch job causes MACRO to create a listing file named ORION.LIS.

3 $ MACRO ALPHA/LIST+MYLIB/LIBRARY-
_$ + [TEST]OLDLIB/LIBRARY + []BETA
$ PRINT ALPHA

> MACRO concatenates the files ALPHA.MAR and BETA.MAR to produce an object file named ALPHA.OBJ and a listing file named ALPHA.LIS. The command line requests the search of libraries MYLIB.MLB (in the current default directory) and OLDLIB.MLB (in the directory [TEST]) for macro definitions. When macro calls are found in BETA.MAR, MACRO searches the libraries OLDLIB, MYLIB, and the system library STARLET.MLB, in that order, for the definitions.
>
> The PRINT command prints the listing file ALPHA.LIS.

4 $ MACRO DELTA+TESTLIB/LIBRARY, ALPHA+MYLIB/LIBRARY

> MACRO requests two separate assemblies. MACRO searches TESTLIB.MLB and the system library STARLET.MLB for macro definitions when macro calls are found in DELTA.MAR, and searches MYLIB.MLB and the system library STARLET.MLB for macro definitions when macro calls are found in ALPHA.MAR.

MAIL

Invokes the Mail Utility, which is used to send messages to other users of the system. For a complete description of the Mail Utility, see the *VMS Mail Utility Manual.*

FORMAT   **MAIL**   *[filespec] [recipient-name]*

MERGE

Invokes the Sort/Merge Utility, which combines 2 to 10 similarly sorted input files and creates a single output file. Note that input files to be merged must be in sorted order. For a complete description of the Sort/Merge Utility, see the *VMS Sort/Merge Utility Manual*.

FORMAT   **MERGE**   *input-filespec1,input-filespec2[,...]*
                          output-filespec

MESSAGE

Invokes the Message Utility, which compiles one or more files of message
definitions. For a complete description of the Message Utility, see the *VMS
Message Utility Manual.*

FORMAT   **MESSAGE**   *filespec[,...]*

MONITOR

Invokes the Monitor Utility, which monitors classes of systemwide performance data at a specified interval. For a complete description of the Monitor Utility, see the *VMS Monitor Utility Manual.*

FORMAT   **MONITOR**   *[class-name[,...]]*

MOUNT

Invokes the Mount Utility, which makes a disk or magnetic tape volume available for processing. For a complete description of the Mount Utility, see the *VMS Mount Utility Manual*.

FORMAT   **MOUNT**   *device-name[:][,...] [volume-label[,...]]*
[logical-name[:]]

Index

Index

D

Index

Index

Index

G

H

I

Index

J

Job
    defining default CPU time limit • DCL1–253,
        DCL2–204, DCL2–347
    defining maximum CPU time limit • DCL1–253,
        DCL2–205, DCL2–348
    deleting from queue • DCL1–136, DCL1–143
    redirecting to another queue • DCL1–46
    removing from queue
        with ASSIGN/MERGE command • DCL1–46
Job batch card
    end of • DCL1–208
JOB card
    password • DCL2–9
JOB command • DCL1–266 to DCL1–271
Job logical name table
    canceling entries • DCL1–105
    including logical name • DCL1–41, DCL1–116
Journal
    EVE • DCL1–195, DCL1–198
    VAXTPU • DCL1–195, DCL1–198

K

Keypad
    application
        setting for DECterm • DCL1–98

L

Label
    command interpreter rules for • DCL1–52,
        DCL1–227, DCL1–229
    in command procedure • DCL1–52, DCL1–227,
        DCL1–229
        syntax • DCL1–227, DCL1–229
    specifying for volume • DCL2–241
    volume header • DCL1–240
    writing on volume • DCL1–240
Lexical functions • DCL1–273, DCL1–274
    F$CONTEXT • DCL1–275
    F$CSID • DCL1–280
    F$CVSI • DCL1–282
    F$CVTIME • DCL1–284

Lexical functions (Cont.)
    F$CVUI • DCL1–286
    F$DEVICE • DCL1–287
    F$DIRECTORY • DCL1–289
    F$EDIT • DCL1–290
    F$ELEMENT • DCL1–292
    F$ENVIRONMENT • DCL1–294
    F$EXTRACT • DCL1–297
    F$FAO • DCL1–299
    F$FILE_ATTRIBUTES • DCL1–306
    F$GETDVI • DCL1–309
    F$GETJPI • DCL1–322
    F$GETQUI • DCL1–328
    F$GETSYI • DCL1–344
    F$IDENTIFIER • DCL1–349
    F$INTEGER • DCL1–351
    F$LENGTH • DCL1–352
    F$LOCATE • DCL1–353
    F$MESSAGE • DCL1–355
    F$MODE • DCL1–356
    F$PARSE • DCL1–358
    F$PID • DCL1–361
    F$PRIVILEGE • DCL1–363
    F$PROCESS • DCL1–364
    F$SEARCH • DCL1–365
    F$SETPRV • DCL1–368
    F$STRING • DCL1–372
    F$TIME • DCL1–373
    F$TRNLNM • DCL1–374
    F$TYPE • DCL1–378
    F$USER • DCL1–380
    F$VERIFY • DCL1–381
    overview • DCL1–272
Library
    object module • DCL1–28
LIBRARY command • DCL1–383
LICENSE command • DCL1–384
License Management Utility (LICENSE) • DCL2–285
Licenses
    displaying active • DCL2–285
Limit working set
    displaying • DCL2–335
LINK command • DCL1–385 to DCL1–391
Linker
    memory allocation file • DCL1–386, DCL1–387
Link option specification records
    analyzing in object file • DCL1–28
List files
    in directory • DCL1–159
Local symbol • DCL1–1, DCL1–5

Index–10

Index

R

Read check
  with APPEND command • DCL1-37
  with COPY command • DCL1-79
  with INITIALIZE command • DCL1-242
READ command • DCL2-28 to DCL2-31
  See also OPEN command
  See also WRITE command
Ready access
  for directories on disk • DCL1-241
RECALL command • DCL2-32 to DCL2-33
Record Management Services
  See RMS
Records
  analyzing
    debugger information • DCL1-27
    end-of-file • DCL1-27
    global symbol directory • DCL1-27
    link option specification • DCL1-28
    module header • DCL1-28
    module traceback • DCL1-28
    patch text • DCL1-24
    relocation • DCL1-29
  comparing • DCL1-151
  reading • DCL2-28
  writing to file • DCL2-399
Recover
  for EDT • DCL1-182
Recovery from system failure
  EVE • DCL1-198
  VAXTPU • DCL1-198
Relocation records
  analyzing in object file • DCL1-29
RENAME command • DCL2-34 to DCL2-37
REPLY command • DCL2-38 to DCL2-46
  See also INITIALIZE command
  See also MOUNT command
  See also REQUEST command
  disabling operator status • DCL2-40
  enabling operator status • DCL2-40
  responding to file system requests • DCL2-39
  responding to user requests • DCL2-39
REQUEST command • DCL2-47 to DCL2-48
Resume execution
  of command procedure • DCL1-63
  of DCL commands • DCL1-63
  of program • DCL1-63

RETURN command • DCL2-49 to DCL2-50
Return key
  pressing to log in • DCL1-392
Rights list
  modifying • DCL2-212
RMS (VMS Record Management Services)
  displaying default block count • DCL2-318
  modifying defaults for • DCL2-214
RUN (Image) command • DCL2-51 to DCL2-52
  abbreviating • DCL2-51
  and debugger • DCL2-51
RUN (Process) command • DCL2-53 to DCL2-62
  See also ATTACH command
  See also SPAWN command
  creating detached process • DCL2-61
Runaway magnetic tape
  stopping • DCL1-241
Runoff
  See DSR
RUNOFF command • DCL2-63 to DCL2-72
RUNOFF/CONTENTS command • DCL2-73 to DCL2-76
RUNOFF/INDEX command • DCL2-77 to DCL2-80

S

Screen-oriented editor • DCL1-180
  EVE • DCL1-189
  VAXTPU • DCL1-189
SEARCH command • DCL2-81 to DCL2-87
Search list • DCL1-39, DCL1-114
Secondary processor
  showing state • DCL2-258
  starting • DCL2-343
  stopping • DCL2-361
Server queue • DCL1-251
SET ACCOUNTING command • DCL2-91 to DCL2-92
  See also ACCOUNTING command
SET ACL command • DCL2-93 to DCL2-99
SET AUDIT command • DCL2-100 to DCL2-111
SET BROADCAST command • DCL2-112 to DCL2-113
SET CARD_READER command • DCL2-114
SET CLUSTER/EXPECTED_VOTES command • DCL2-115 to DCL2-116
SET command • DCL2-88 to DCL2-90
  summary of options • DCL2-88
SET COMMAND command • DCL2-117
SET CONTROL command • DCL2-118 to DCL2-119

Index

Index

T

Index

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-DEC-DEMO (800-332-3366) using a 1200- or 2400-baud modem. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

| Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	———	Local Digital subsidiary or approved distributor
Internal[1]	———	USASSB Order Processing - WMO/E15 or U.S. Area Software Supply Business Digital Equipment Corporation Westminster, Massachusetts 01473

[1]For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

_____

What I like best about this manual is _____

_____

_____

What I like least about this manual is _____

_____

_____

I found the following errors in this manual:

Page   Description

_____   _____

_____   _____

_____   _____

_____   _____

_____   _____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

digital™

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01–3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

Reader's Comments

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

_____

What I like best about this manual is _____

_____

_____

What I like least about this manual is _____

_____

_____

I found the following errors in this manual:

Page   Description

____   _____

____   _____

____   _____

____   _____

____   _____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

d i g i t a l™

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01–3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987