VMS Software

VSI OpenVMS

DCL Dictionary: A–M

Operating System and Version: VSI OpenVMS IA-64 Version 8.4-1H1 or higher
VSI OpenVMS Alpha Version 8.4-2L1 or higher
VSI OpenVMS x86-64 Version 9.2-1 or higher

VMS Software, Inc. (VSI)
Boston, Massachusetts, USA

DCL Dictionary: A–M

VMS Software

Legal Notice

Table of Contents

Preface

1. About VSI

VMS Software, Inc. (VSI) is an independent software company licensed by Hewlett Packard Enterprise to develop and support the OpenVMS operating system.

2. Intended Audience

This manual is intended for all users of the VSI OpenVMS 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 *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/].

3. Documents Structure

This manual contains detailed descriptions of each command and lexical function. The commands are listed in alphabetical order. The lexical functions are grouped under the heading Lexical Functions and are listed alphabetically within that grouping.

The hardcopy version of the *VSI OpenVMS DCL Dictionary* is a two-part manual. The first volume contains commands beginning with the letters A to M (including the lexical functions); the second volume contains commands beginning with the letters N to Z.

Appendix A of this manual (in the second volume of the hardcopy manual) lists the obsolete DCL commands and the current services that replace them.

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

4. Related Documents

For an introduction to the OpenVMS operating system and for information on using DCL, refer to the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/]. This manual is especially recommended for novice users or users lacking experience with interactive computer systems.

The *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/] provides an overview of DCL command language concepts and defines and illustrates good practices in constructing command procedures with DCL commands and lexical functions.

Refer to the various utilities reference manuals for detailed information about 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. The *VSI OpenVMS DCL Dictionary* provides only a brief description and format information for each utility.

For message descriptions, use the online Help Message utility.

5. VSI Encourages Your Comments

You may send comments or suggestions regarding this manual or any VSI document by sending electronic mail to the following Internet address: <docinfo@vmssoftware.com>. Users who have VSI OpenVMS support contracts through VSI can contact <support@vmssoftware.com> for help with this product.

6. OpenVMS Documentation

The full VSI OpenVMS documentation set can be found on the VMS Software Documentation webpage at https://docs.vmssoftware.com.

7. Conventions

The following conventions are used in this manual:

Convention	Meaning
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** and then press and release another key or a pointing device button.
. . .	A horizontal ellipsis in examples 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 command format descriptions, parentheses indicate that you must enclose choices in parentheses if you specify more than one.
[ ]	In command format descriptions, brackets indicate optional choices. You can choose one or more items or no items. Do not type the brackets on the command line. However, you must include the brackets in the syntax for OpenVMS directory specifications and for a substring specification in an assignment statement.
|	In command format descriptions, vertical bars separate choices within brackets or braces. Within brackets, the choices are optional; within braces, at least one choice is required. Do not type the vertical bars on the command line.
{ }	In command format descriptions, braces indicate required choices; you must choose at least one of the items listed. Do not type the braces on the command line.
bold type	Bold type represents the name of an argument, an attribute, or a reason. Bold type also represents the introduction of a new term.

Convention	Meaning
italic type	Italic type indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error *number*), in command lines (**/PRODUCER=*name***), and in command parameters in text (where *dd* represents the predefined code for the device type).
UPPERCASE TYPE	Uppercase type indicates the name of a routine, the name of a file, or the abbreviation for a system privilege.
`monospace type`	Monospace type indicates code examples, command examples, and interactive screen displays. In text, this type also identifies URLs, UNIX commands and pathnames, PC-based commands and folders, and certain elements of the C programming language.
`bold monospace type`	Bold monospace type indicates a DCL command or command qualifier.
-	A hyphen at the end of a command format description, command line, or code line indicates that the command or statement continues on the following line.
numbers	All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radixes – binary, octal, or hexadecimal – are explicitly indicated.

DCL Commands

! (Comment Delimiter)

! (Comment Delimiter) — Indicates that everything that follows it on a command line is a comment and should not be processed as part of a command.

Format

! *comment-text*

Example

```
$ !
$ WRITE SYS$OUTPUT "hello"     ! This command should output "hello".
hello
$ FOO = " "                    ! This command defines FOO as a blank.
$ FOO WRITE SYS$OUTPUT "hello" ! This command should output "hello".
hello
$ FOO = "!"                    ! This command defines FOO as a !.
$ FOO WRITE SYS$OUTPUT "hello" ! This command should be ignored.
$ ! WRITE SYS$OUTPUT "hello"   ! This command should be ignored too.
```

= (Assignment Statement)

= (Assignment Statement) — Defines a symbolic name for a character string or integer value.

Synopsis

symbol-name* =[=] *expression

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

---

Note

VSI advises against assigning a symbolic name that is already a DCL command name. VSI especially discourages the assignment of symbols such as IF, THEN, ELSE, and GOTO, which can affect the interpretation of command procedures.

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 relevant sections in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#SYMBOLS_CH].

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.

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 system 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 ALAMO.TXT BEST.DOC
```

   The files `ALAMO.TXT` and `BEST.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`.

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.

6.
```
$ $=34
%DCL-W-NOCOMD, no command on line - reenter with alphabetic first
character
$ $$=34
$ SHOW SYMBOL $$
%DCL-W-UNDSYM, undefined symbol - check validity and spelling
$ SHOW SYMBOL $
$ = 34   Hex = 00000022  Octal = 00000000042
```

If you begin a symbol name with the dollar sign ($), use two dollar signs ($$) because DCL discards the first instance of the dollar sign.

7.
```
$ COUNT = 0
$ LOOP:
$       COUNT = COUNT + 1
$       IF P'COUNT' .EQS. "" THEN EXIT
$       APPEND/NEW &P'COUNT' SAVE.ALL
$       DELETE &P'COUNT';*
$       IF COUNT .LT. 16 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 sixteen file names can be passed to the command procedure. The file names are assigned to the symbols P1, P2, and so on. This is applicable only when you set bit 3 of DCL_CTLFLAGS to 1.

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 ALAMO.TXT BEST.DOC
```

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

:= (String Assignment)

:= (String Assignment) — Defines a symbolic name for a character string value.

Format

symbol-name* :=[=] *string

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

Note

VSI advises against assigning a symbolic name that is already a DCL command name. VSI especially discourages the assignment of symbols such as IF, THEN, ELSE, and GOTO, which can affect the interpretation of command procedures.

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.

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 character (-) as a continuation character. For example:

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

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 relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#SYMBOL_SUBSTITUTION] for more information on symbol substitution.

You can also use the string assignment statement to define a foreign command. See the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#SYMBOLS_CH] 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 *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-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 relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#SYMBOL_SUBSTITUTION].

Examples

1.
```
$ TIME := SHOW TIME
$ TIME
24-DEC-2001 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 := $DKA1:[TEDESCO]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.

   When you subsequently enter 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.

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)

@ (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. The asterisk (*) and the percent sign (%) 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.

Setting bit 3 of DCL_CTLFLAGS to 1, specifies from one to sixteen optional parameters to pass to the command procedure. The symbols (P1, P2, … P16) are assigned character string values in the order

of entry. If you clear the bit 3 of DCL_CTLFLAGS, the default parameters are set (that is, (P1, P2, … P8)).

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
  ```

  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 32, 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
  14-SEP-2001 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          14-DEC-2001 15:47:45.57
FEBRUARY.TXT;7         14-DEC-2001 15:43:16.20
MARCH.TXT;6            14-DEC-2001 11:11:45.74
   .
   .
   .
Total of 11 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.

The asterisk (*) and the percent sign (%) 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.

   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. The entire file is treated by DCL as command input. 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/TPU 'NAME'.DOC
$       GOTO NEXT
Ctrl/Z
$ @EDOC
```

This procedure, named EDOC.COM, invokes the EVE 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 .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$INPUT for the duration of the procedure. This change allows the EVE editor to read input data from the terminal, rather than from the command procedure file (the default input data stream if SYS$INPUT had not been changed). When the command procedure exits, SYS$INPUT is reassigned to its original value.

7.
```
! PEOPLE.DAT
! A set of data with embedded key qualifiers for the SORT command.
!
! Usage: SORT@PEOPLE.DAT
!
/KEY=(POS:10,SIZE:10) sys$input people.out
Fred     Flintstone    555-1234
Barney   Rubble        555-2244
Wilma    Flintstone    555-1234
Betty    Rubble        555-2244
```

```
George    Slate          555-8911
Dino      Dinosaur       555-1234
$!
$ SORT@PEOPLE.DAT
$ purge people.out
$ type people.out
```

Creates a sorted list of people in file PEOPLE.OUT and displays it. This demonstrates when using "@" in the middle of a DCL command, DCL redirects the entire file as command input.

8.
```
$ 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' 'P9'
  'P10' 'P11' 'P12'  'P13'  'P14'  'P15'  'P16'
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 P16 can be used to specify additional qualifiers. This is applicable if bit 3 of DCL_CTLFAGS is set to 1. 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.

ACCOUNTING

ACCOUNTING — Runs the Accounting utility, which produces reports of resource use.

Format

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

Description

For more information about the Accounting utility, see the relevant section in the *VSI OpenVMS System Management Utilities Reference Manual, Volume 1: A-L* [https://docs.vmssoftware.com/vsi-openvms-system-management-utilities-reference-manual-volume-i-a-l/#ACC_U].

ALLOCATE

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. Requires read (R), write (W), or control access.

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. To determine which devices apply to specific OpenVMS versions, see SPD.

Devices by Classification				
Disk Devices				
EF51	EF52	EF53	EF54	EF58
ESE20	ESE25	ESE52	ESE56	ESE58
EZ31	EZ31L	EZ32	EZ32L	EZ33
EZ33L	EZ34	EZ35	EZ51	EZ52
EZ53	EZ54	EZ56R	EZ58	HSZ10
HSZ15	HSZ20	HSZ40	ML11	RA60
RA70	RA71	RA72	RA73	RA80
RA81	RA82	RA90	RA92	RAH72
RB02	RB80	RC25	RCF25	RD26
RD31	RD32	RD33	RD51	RD52
RD53	RD54	RF30	RF31	RF31F
RF32	RF35	RF36	RF37	RF70
RF71	RF72	RF73	RF74	RF75
RFF31	RFH31	RFH32	RFH35	RFH72

Devices by Classification				
RFH73	RK06	RK07	RL01	RL02
RM03	RM05	RM80	RP04	RP05
RP06	RP07	RP07HT	RX01	RX02
RX04	RX18	RX23	RX23S	RX26
RX33	RX33S	RX35	RX50	RZ01
RZ13	RZ14	RZ15	RZ16	RZ17
RZ18	RZ22	RZ23	RZ23L	RZ24
RZ24L	RZ25	RZ25L	RZ26	RZ26B
RZ26L	RZ26M	RZ27	RZ27B	RZ27L
RZ28	RZ28B	RZ28L	RZ29	RZ29B
RZ31	RZ34L	RZ35	RZ35L	RZ36
RZ36L	RZ37	RZ38	RZ55	RZ55L
RZ56	RZ56L	RZ57	RZ57I	RZ57L
RZ58	RZ59	RZ72	RZ73	RZ73B
RZ74	RZ74B	RZ75	RZ75B	RZF01
Compact Disk Devices				
RRD40	RRD40S	RRD42	RRD43	RRD44
RRD50	RV20	RV60	RV80	RW504
RW510	RW514	RW516	RWZ01	RWZ21
RWZ31	RWZ51	RWZ52	RWZ53	RWZ54
Tape Devices				
TA78	TA79	TA81	TA85	TA86
TA87	TA90	TA90E	TA91	TAD85
TAPE9	TD34	TD44	TE16	TF30
TF70	TF85	TF86	TK50	TK50S
TK60	TK70	TK70L	TKZ09	TKZ60
TL810	TL820	TLZ04	TLZ06	TLZ07
TLZ6	TLZ7	TM32	TS11	TSZ05
TSZ07	TSZ08	TU45	TU56	TU58
TU77	TU78	TU80	TU81	TZ30
TZ30S	TZ85	TZ857	TZ86	TZ865
TZ867	TZ87	TZ875	TZ877	TZ88
TZ885	TZ887	TZ89	TZ895	TZ897
TZK10	TZK11	TZX0		

/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 diskette drive and makes its name equivalent to the process logical name ACCOUNTS.

ANALYZE/AUDIT

ANALYZE/AUDIT — Invokes the Audit Analysis utility, which selectively extracts and displays information from security audit log files or security archive files.

Format

ANALYZE/AUDIT[*filespec*]

Description

For more information about the Audit Analysis utility, see the relevant sections in the *VSI OpenVMS System Management Utilities Reference Manual, Volume 1: A-L* [https://docs.vmssoftware.com/vsi-openvms-system-management-utilities-reference-manual-volume-i-a-l/#AUDITING_PART] or online help.

ANALYZE/CRASH_DUMP

ANALYZE/CRASH_DUMP — Invokes the System Dump Analyzer utility, which analyzes a system dump file. The **/CRASH_DUMP** qualifier is required.

Format

ANALYZE/CRASH_DUMP*filespec*

Description

Invokes the System Dump Analyzer utility, which analyzes a system dump file. The **/CRASH_DUMP** qualifier is required. For more information about the System Dump Analyzer utility on Alpha, refer to the relevant section in the *VSI OpenVMS System Analysis Tools Manual* [https://docs.vmssoftware.com/vsi-openvms-system-analysis-tools-manual/#PART1] or online help.

For OpenVMS Alpha Systems

You can also use the **ANALYZE/CRASH_DUMP** command with process dumps. However, the preferred command is **ANALYZE/PROCESS**, which provides complete access to the information in the dump.

ANALYZE/DISK_STRUCTURE

ANALYZE/DISK_STRUCTURE — Invokes the Analyze/Disk_Structure utility. Checks the readability and validity of Files-11 On-Disk Structure Level 1, 2, and 5 disk volumes. Reports errors and inconsistencies.

Format

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

Description

The **/DISK_STRUCTURE** qualifier is required.

For more information about the Analyze/Disk_Structure utility, see the relevant section in the *VSI OpenVMS System Management Utilities Reference Manual, Volume 1: A-L* [https://docs.vmssoftware.com/vsi-openvms-system-management-utilities-reference-manual-volume-i-a-l/#VERI_U] or online help.

Qualifiers

/EXTENTS

Produces a report on the fragmentation of free space on a volume. By default, the only output is the number of extents of free space and the total number of free blocks. For additional details, specify one of the following additional qualifiers with the **ANALYZE/DISK_STRUCTURE/EXTENTS** command:

Qualifier	Syntax	Description
/LARGEST	/LARGEST[=*n*]	Displays a list of block counts for the *n* largest extents of free space on the volume in descending size order. The default for *n* is 10. The qualifier is ignored if you specify zero or a negative number. The list is also saved in the DCL symbol ANALYZE$LARGEST_EXTENTS (as a comma-separated list of decimal values). The symbol is set to an empty string if there is no free space on the disk. There is no upper limit on *n*, but if the DCL symbol exceeds 1024 characters, the number of extents in the symbol will be reduced to ensure the symbol is no more than 1024 characters.
/LOCK_VOLUME	/LOCK_VOLUME /NOLOCK_VOLUME	Locks the volume against allocation while the data is being collected. By default, the volume is locked.
/OUTPUT	/OUTPUT[=*filespec*] /NOOUTPUT	Specifies the output file for the fragmentation report produced by the ANALYZE/DISK_STRUCTURE utility. If you omit the qualifier or the entire filename, SYS$OUTPUT will be used. The default filename is ANALYZE$EXTENTS.LIS.
/REQUIRED	/REQUIRED=n	Displays the number of extents required to satisfy an allocation request of *n* blocks (starting with the largest extent). There is no default for *n*. If you specify zero or a negative number, the qualifier is ignored. The result is also saved in the DCL symbol ANALYZE$REQUIRED_EXTENTS. The symbol is set to an empty string if there is insufficient space on the disk to satisfy the allocation request.

Example:

```
$ ANALYZE/DISK_STRUCTURE/EXTENTS/LARGEST/REQUIRED=20000 LDM9063:

Extent report for _X86VMS$LDM9063:
```

```
================================

The disk has 6 extents of free space for a total of 25262 free blocks.

The extent sizes are:

             17176
              5591
              2469
                15
                 9
                 2

2 extents are required for an allocation of 20000 blocks.
```

ANALYZE/ERROR_LOG/ELV (Alpha/Integrity servers Only)

ANALYZE/ERROR_LOG/ELV (Alpha/Integrity servers Only) — Invokes the Error Log Viewer (ELV) to selectively report the contents of one or more error log files. This utility is most useful with error logs written on systems running OpenVMS Version 7.3 and later. For more information about the Error Log Viewer, see the relevant section in the *VSI OpenVMS System Management Utilities Reference Manual, Volume 1: A-L* [https://docs.vmssoftware.com/vsi-openvms-system-management-utilities-reference-manual-volume-i-a-l/#ELV_CHAP] or online help.

Format

ANALYZE/ERROR_LOG/ELV [*command*]

Description

For error logs written on OpenVMS Version 7.2* systems, you must use the **DIAGNOSE** command, which invokes the DEC event utility. The related documentation is available online.

ANALYZE/IMAGE

ANALYZE/IMAGE — Analyzes the contents of an executable image file or a shareable image file on OpenVMS Alpha systems, and an Executable and Linkable Format (ELF) image file or sharable image file on OpenVMS Integrity server systems, identifying obvious errors in the file. This analysis includes translated images on Integrity servers and Alpha systems. The **/IMAGE** qualifier is required. For general information about image files, see the description of the linker in the *VSI OpenVMS Linker Utility Manual* [https://docs.vmssoftware.com/vsi-openvms-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.

The asterisk (*) and percent sign (%) wildcard characters 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 on OpenVMS Alpha systems, and of an Executable and Linkable Format (ELF) image file or sharable image file on OpenVMS Integrity server systems. 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.

On OpenVMS Integrity server systems, the **ANALYZE/IMAGE** command automatically distinguishes between Integrity servers and Alpha images by examining the header information.

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.

Notes

For Integrity servers images and objects, the Analyze utility determines whether the file it analyzes is an image file or object file. Although Analyze allows you to specify **ANALYZE/OJBECT** on an ELF image file, use **ANALYZE/IMAGE** for ELF image files and **ANALYZE/OJBECT** for ELF object files.

When parsing output from **ANALYZE/IMAGE**, be aware that the output for ELF images may change.

When using **ANALYZE** without a qualifier, the default is **/OBJECT**. Therefore, when using this default to analyze an image in the output file, the utility correctly identifies itself as"Analyze Object File".

The OpenVMS Alpha version of **ANALYZE/IMAGE** does not have the capability of analyzing all non-platform images.

When you analyze Integrity servers images on IA-64 platforms, **ANALYZE/IMAGE** accepts Alpha-only qualifiers, but ignores any effect of these qualifiers.

Depending on the platform, the **ANALYZE/IMAGE** command distinguishes Integrity servers images from Alpha images by examining the meta information (for example, ELF, EIHD, or IHD).

The **ANALYZE/IMAGE** command provides the following information for image files:

Image architecture and type	The OpenVMS platform and whether the image is executable or shareable.
Image name	The name of the image or shareable image.
Image identification	The identification given in a link operation.
Creating linker identification	The linker that generated the image.
Link date and time	The date and time of the link operation.
Image transfer addresses	The addresses to which control is passed at image execution time.
Image version	The revision level (major ID and minor ID) of the image.

Location and size of the image's symbol vector (Alpha and Integrity servers only).	
List of required sharable images	The dependencies on sharable images.
Location of the debugger symbol table (DST)	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 (Alpha only).
Location and interpretation of the debug and traceback information	The sections that contain the information and formats the data (DWARF) (Integrity servers only).
Location of the global symbol table (GST)	The location of the GST in the image file. GST information is present only in shareable image files (Alpha only).
Location of the global symbol table (.symtab)	The location of the GST in the image file. GST information is present only in shareable image files (Integrity servers only).
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 (Alpha only).
Image section descriptors (ISD)	Identify portions of the image binary contents that are grouped in OpenVMS Cluster systems 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.
Summary of internal tables	Lists the program segments and sections of which the image consists (Integrity servers only).
Fixup vectors	Contain information that the image activator needs to ensure the position independence of shareable image references (Alpha only).
Fixup information	Information that the image activator needs to ensure the position independence of shareable image references (Integrity servers only).
System version categories	For an image that is linked against the executive (the system shareable image on Integrity servers and Alpha), 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. For Alpha images, 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.

The default behavior for analyzing ELF images differs from the behavior for analyzing Alpha images. For ELF images, a summary of the major ELF tables is displayed. With this information, you can select specific segments and/or sections for analysis. To locate errors, analyze the entire image by selecting all sections and segments.

Qualifiers

/FIXUP_SECTION (Alpha only)

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.

/FLAGVALUES=(keyword[,...]) (Integrity servers only)

Several fields in an ELF module represent bit flags. Where possible, these bit-flag values are examined and displayed individually. By default, only the flag values that are set to 1 (ON) are displayed.

The keywords are as follows:

Keyword	Description
ON	The keyword ON displays all flags whose value is 1.
OFF	The keyword OFF displays all flags whose value is 0.
ALL	The keyword ALL displays all flag values. The keywords ON and OFF, in contrast, indicate the value of each specific flag bit.

/GST (Alpha only)

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 (Alpha only)

Positional qualifier.

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

/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.

/MODULE [=(*module_name*[,...])] (Integrity servers only)

Selectively formats debug or traceback information for the named module or list of modules. You must request debug or traceback information by using the **/SECTIONS** qualifier with keywords ALL, DEBUG, or TRACE. If debug or traceback information is selectively formatted, then the module name is a subselection.

If you do not specify a module name, only debug or traceback meta information about the available modules is printed. In this case, any other debug or traceback selection is deactivated.

Note

This qualifier is only valid for **ANALYZE/IMAGE**. Although **ANALYZE/OBJECT** can be used to format Integrity servers images, Analyze rejects the **/MODULE** qualifier.

/OUTPUT=*filespec*

Identifies the output file for storing the results of the image analysis. The asterisk (*) and the percent sign (%) wildcard characters are not 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.

/PAGE_BREAK=*keyword* (Integrity servers only)

Specifies if and where page breaks (form feeds) are inserted in the report file. This qualifier is only useful if **/OUTPUT** is used to write a report file. It is ignored if **/INTERACTIVE** is used to specify an interactive analysis.

The keywords are as follows:

Keyword	Description
NONE	Creates a report without any page break.
PRINTABLE_REPORT	Creates a printable report with page breaks as in listing files. The number of lines per page is the default number of lines on a printer page. This is the default behavior for ANALYZE_IMAGE when no qualifier is specified.
SEPARATE_INFORMATION	Inserts a page break between different section information.

/SECTIONS [=(*keyword*[,...])] (Integrity servers only)

Selects individual program sections or section types to display.

Note

This qualifier and its keywords can only be used to form an inclusion list of sections to be displayed. This qualifier is not negatable and cannot be used to form an exclusion list. If no values are specified, the default keyword is HEADERS.

The keywords are as follows:

Keyword	Description
ALL	Displays a detailed analysis of every section in the module. Note that this keyword can generate a large amount of output.
CODE	Displays all of all sections of type SHT_PROGBITS where the executable flag is set (SHDR$M_SHF_EXECINSTR in the section header). The section data will be displayed as machine instructions.
DEBUG [=(suffix[,...])]	Analyzes and displays sections consisting of debug information. In addition, you can use a list of debug section name suffixes to selectively format DEBUG information. The debug section names, which appear as ".debug_suffix", can be viewed in the summary table. The suffix can be specified as follows: ● ABBREV -- Format DEBUG abbreviations ● ARANGES -- Formats DEBUG address lookup tables ● FRAME -- Formats DEBUG frame descriptors for unwinding ● INFO -- Formats DEBUG symbols ● LINE -- Formats DEBUG source line info ● PUBNAMES -- Formats DEBUG name lookup tables ● PUBTYPES -- Formats DEBUG type lookup tables
EXTENSIONS	Analyzes and displays sections of type SHT_IA64_EXT. The data is displayed in hexadecimal format.
GROUP	Analyzes and displays sections of type SHT_GROUP. Sections of this type consist of a list of the section numbers of sections belonging to that group.
HEADERS	The default keyword. Displays the ELF header and the section header details.
LINKAGES	Analyzes and displays sections of type SHT_VMS_LINKAGES. The data is displayed as a list of linkage descriptors.
NOBITS	Analyzes and displays sections of type SHT_NOBITS. There is no module data associated with sections of this type.
NOTE	Analyzes and displays sections of type SHT_NOTE. The data for this section is displayed as a list of formatted OpenVMS note entries.
NULL	Displays all sections of type PT_NULL. No data will be displayed for segments of this type.
NUMBERS= (number [,...])	Displays individual sections, as follows: ● The selected sections will have a detailed display of their header and their contents. An informational message is

Keyword	Description
	displayed for section numbers that do not exist in the module. • One or more numeric values may be specified. • Section numbers may be specified in decimal, octal (using the %O prefix), or hexadecimal (using the %X prefix).
STRTAB	Analyzes and displays sections of type SHT_STRTAB. The data for this section is displayed as a string table.
SYMTAB	Displays sections of type SHT_SYMTAB. The data for this section is displayed as a symbol table.
SYMBOL_VECTOR	Sections of this type will only appear in sharable image files. If present, they point to the same data as the dynamic segment DT_VMS_SYMVEC tags.
TRACE [=(suffix[,...])]	Analyzes and displays sections consisting of traceback information. In addition, you can use a list of trace section name suffixes to selectively format TRACE information. The trace section names, which appear as ".trace_suffix", can be viewed in the summary table. The suffix can be specified as shown below. In addition, because there is one common debug and traceback section, ".debug_line", the suffix "line" can be specified as shown below as well: • ABBREV -- Formats TRACE abbreviations • ARANGES -- Formats TRACE address lookup tables • INFO -- Formats TRACE symbols • LINE -- Formats TRACE source line info
UNWIND	Analyzes and displays sections of type SHT_IA64_UNWIND. Each section of this type has an associated Unwind Information section of type SHT_PROGBITS. This associated section is also displayed.

/SEGMENTS [=(*keyword*[,...])] (Integrity servers only)

Selects individual program segments or program segments of a specified type to be displayed.

Note

This qualifier and its keywords can only be used to form an inclusion list of segments to be displayed. This qualifier is not negatable and cannot be used to form an exclusion list. If no values are specified, the default keyword is HEADERS.

The keywords are as follows:

Keyword	Description
ALL	Analyzes and displays information for every program segment. Note that this can generate a large amount of output.
CODE	Analyzes and displays all executable segments (PHDR$M_PF_X bit set in the segment header). Segment data is displayed as machine instructions.
DYNAMIC	Analyzes and displays the segment of type PT_DYNAMIC.
EXTENSIONS	Analyzes and displays segments of type IA_64_ARCHEXT.
HEADERS	The default keyword. Analyzes and displays the ELF header and segment header details.
LOAD	Analyzes and displays segments of type PT_LOAD. If the segment header indicates this is an executable segment (PHDR$M_PF_X bit set in the segment header), the contents will be formatted as machine instructions, otherwise the contents are formatted as hexadecimal data.
NULL	Analyzes and displays segments of type PT_NULL. No a data will be displayed for segments of this type.
NUMBERS= (number [,...])	Analyzes and displays individual segments, as follows: ● The selected segments have a detailed display of header and content information. For section numbers that do not exist in the module, an informational message is displayed. ● One or more numeric values may be specified. ● Segment numbers may be specified in decimal, octal (using the %O prefix), or hexadecimal (using the %X prefix).

/SELECT=(*keyword*[,...])

Allows for the collection of specific image file information and displays the selected keyword items in the order specified.

Analyze creates DCL symbols for all selectable information with the **/SELECT** qualifier. The symbol names consist of the prefix ANALYZE$ and a descriptive name of the information they hold. The symbol value is the selected information, usually printed to SYS$OUTPUT. Effectively, all of the printed information is duplicated in the symbols. For unselected information, the corresponding symbols will contain the null string.

The keywords are as follows:

Keyword	Description
ARCHITECTURE	Writes the architecture information into the DCL symbol ANALYZE$ARCHITECTURE. Returns "OpenVMS IA64" if the file is an OpenVMS Integrity servers image file. Returns an OpenVMS Alpha image file."OpenVMS Alpha" if the file is an OpenVMS Alpha image file.
BUILD_IDENTIFICATION	Writes build identification information into the DCL symbol ANALYZE$BUILD_IDENTIFICATION. For OpenVMS

Keyword	Description
	Integrity servers and Alpha image files, returns the image build identification stored in the image file, enclosed in quotation marks.
FILE_TYPE	Writes file type information into the DCL symbol ANALYZE$FILE_TYPE. Returns "Image" if the file is an OpenVMS Integrity servers or Alpha image file.
IDENTIFICATION [=keyword]	The possible keywords are as follows: • IMAGE (default) --- Writes the image identification information into the DCL symbol ANALYZE $IDENTIFICATION. Returns the image identification that is stored in the image file, enclosed in quotation marks. Otherwise, returns "Unknown". • LINKER --- Writes the linker identification information into the DCL symbol ANALYZE$LINKER_IDENTIFICATION. Returns the identification of the linker used to link the image.
IMAGE_TYPE	Writes image type information into the DCL symbol ANALYZE$IMAGE_TYPE. Returns "Shareable" if the file is a shareable image file. Returns "Executable"if the file is either an OpenVMS Integrity servers or Alpha executable (non-shareable) image file.
LINK_TIME	Writes link time information into the DCL symbol ANALYZE$LINK_TIME. Returns the image link time that is stored in the image file, enclosed in quotation marks.
NAME	Writes the image name into the DCL symbol ANALYZE $NAME. For image files, returns the image name that is stored in the image header, enclosed in quotation marks.
VERSION_NUMBERS (Alpha/ Integrity servers only)	If an image depends on the system base image and system components, **ANALYZE** writes the version numbers from the image into DCL symbols. The symbols are named after the components. The symbol values contain the minor and major version numbers. When the image is for the same platform on which **ANALYZE** is running, the version numbers from the running system are also written and compared.

Note

The Analyze utility can work on several files. Because there is only one set of DCL symbols, the symbols only contain information from the last analyzed file. When an error occurs, symbol values are undefined. Check for Analyze errors first, then use the symbols.

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.

2. `$ ANALYZE/IMAGE/OUTPUT=LIALPHEX/FIXUP_SECTION/PATCH_TEXT  LINEDT, ALPRIN`
   `(Alpha only)`

   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 `ALPRIN.EXE` in file `LIALPHEX.ANL`. Output is sent to the file `LIALPHEX.ANL`.

3. `$ ANALYZE/IMAGE/SELECT=(ARCH,FILE,NAME,IDENT,BUILD,LINK) *.EXE`
   `DISK:[DIRECTORY]ALPHA.EXE;1OpenVMS`
   `AlphaImage"MAIL""V1.06""XBCA-0080070002"19-MAR-2008 11:17:50.76`

   On an Alpha system, this example displays the information requested about the executable file ALPHA.EXE.

4. `$ ANALYZE/IMAGE/SELECT=(ARCHITECTURE,IDENT,NAME) HELLO` ❶
   ```
   USER:[JOE]HELLO.EXE;1
   OpenVMS IA64
   "V1.0"
   "HELLO"
   $
   $ SHOW SYMBOL ANALYZE$*
   ANALYZE$ARCHITECTURE = "OpenVMS IA64"
   ANALYZE$BUILD_IDENTIFICATION = ""
   ANALYZE$FILE_TYPE = ""
   ANALYZE$IDENTIFICATION = ""V1.0""
   ANALYZE$IMAGE_TYPE = ""
   ANALYZE$LINKER_IDENTIFICATION = ""
   ANALYZE$LINK_TIME = ""
   ANALYZE$NAME = ""HELLO""
   $
   $ ANALYZE/IMAGE/SELECT=(IDENT=(IMAGE,LINKER),IMAGE,LINK) HELLO ❷
   USER:[JOE]HELLO.EXE;1
   "V1.0"
   "Linker I01-54"
   Executable
   7-JUN-2004 11:47:08.10
   $
   $ SHOW SYMBOL ANALYZE$*
   ANALYZE$ARCHITECTURE = ""
   ANALYZE$BUILD_IDENTIFICATION = ""
   ANALYZE$FILE_TYPE = ""
   ANALYZE$IDENTIFICATION = ""V1.0""
   ANALYZE$IMAGE_TYPE = "Executable"
   ANALYZE$LINKER_IDENTIFICATION = ""Linker I01-54""
   ANALYZE$LINK_TIME = " 7-JUN-2004 11:47:08.10"
   ANALYZE$NAME = ""
   $
   $ ANALYZE/IMAGE/SELECT=FILE HELLO.* ❸
   USER:[JOE]HELLO.C;1
   %ANALYZE-E-ILLFIL, Illegal file format encountered
   USER:[JOE]HELLO.EXE;1
   Image
   USER:[JOE]HELLO.MAP;1
   %ANALYZE-E-ILLFIL, Illegal file format encountered
   USER:[JOE]HELLO.OBJ;1
   Object
   $
   $ SHOW SYMBOL ANALYZE$*
   ```

```
ANALYZE$ARCHITECTURE = ""
ANALYZE$BUILD_IDENTIFICATION = ""
ANALYZE$FILE_TYPE = "Object"
ANALYZE$IDENTIFICATION = ""
ANALYZE$IMAGE_TYPE = ""
ANALYZE$LINKER_IDENTIFICATION = ""
ANALYZE$LINK_TIME = ""
ANALYZE$NAME =
$
```

This Integrity servers example displays the information requested for the executable file, `HELLO.EXE`. The following text is keyed to the callout numbers at the ends of each **`ANALYZE/IMAGE`** command line in the example:

❶ Only the selected information can be found in the DCL symbols. The information in the symbols is identical to what is printed to SYS$OUTPUT, that is, if quoted strings are printed, there are quoted strings in the symbol.

❷ If the new linker identification is selected, it is necessary to use IDENT with a keyword list.

❸ When using wildcards, errors in the analyzed file (for example, illegal file format errors) do not terminate Analyze. Only the information from the last analyzed file can be found in the DCL symbols.

ANALYZE/MEDIA

ANALYZE/MEDIA — Invokes the Bad Block Locator utility, which analyzes block-addressable devices and records the location of blocks that cannot store data reliably. For more information about the Bad Block Locator utility, see the *OpenVMS Bad Block Locator Utility Manual* or online help.

Format

`ANALYZE/MEDIA`*`device`*

ANALYZE/OBJECT

ANALYZE/OBJECT — Analyzes the contents of an object file on OpenVMS Alpha systems, and an Executable and Linkable Format (ELF) object file on OpenVMS Integrity server systems, and identifies obvious errors. The **`/OBJECT`** qualifier is required.

Synopsis

`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. The asterisk (*) and the percent sign (%) wildcard characters are allowed in the file specification.

Description

For general information about object files, see the description of the linker in the *VSI OpenVMS Linker Utility Manual* [https://docs.vmssoftware.com/vsi-openvms-linker-utility-manual/]. Use the **ANALYZE/IMAGE** command to analyze the contents of an image file.

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 all records in an object module conform in content, format, and sequence to the specifications of the Integrity servers or Alpha Object Language.

On OpenVMS Integrity server systems, the **ANALYZE/OBJECT** command automatically distinguishes Integrity servers and Alpha objects by examining the format of the object modules header.

ANALYZE/OBJECT is intended primarily for programmers of compilers, debuggers, or other software involving the operating system's object modules. It checks that the ELF object format (Integrity servers or the object language records (Alpha) generated by the object modules are acceptable to the Linker utility, 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 linker and on the Alpha object languages, see the *VSI OpenVMS Linker Utility Manual* [https://docs.vmssoftware.com/vsi-openvms-linker-utility-manual/].

Notes

For Integrity servers images and objects, the Analyze utility determines whether the file it analyzes is an image file or object file. Although Analyze allows you to specify **ANALYZE/IMAGE** on an ELF object file, use **ANALYZE/IMAGE** for ELF image files and **ANALYZE/OJBECT** for ELF object files.

The OpenVMS Alpha versions of **ANALYZE/OBJECT** are not fully capable of analyzing non-platform objects (for example Integrity servers objects on Alpha).

The output format of **ANALYZE/OBJECT** for ELF objects may change. Further, the default behavior for analyzing ELF objects differs from the behavior for analyzing Alpha objects. For ELF objects, a summary of the major ELF tables is displayed. With this information, you can select specific sections for further analysis. To locate errors, the entire object should be analyzed by selecting all sections.

When you analyze Integrity servers objects on IA-64 platforms, **ANALYZE/OBJECT** accepts Alpha-only qualifiers, but ignores any effect of these qualifiers.

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 OpenVMS Integrity servers or Alpha Object Language. This comparison is particularly important if the analysis output contains a diagnostic message.

ANALYZE/OBJECT displays the following information for object modules:

- Module architecture and type
- Module name
- Module version
- Module creation date and time

- Language processor creator

Linking an object module differs from analyzing an object module. The object's contents are not interpreted; rather, only the meta information is checked for consistency. 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 for Alpha objects:

- 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
```

For Alpha objects, 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

For Alpha objects, End-of-Module (EOM) records and module header (MHD) records are always analyzed, no matter which qualifiers you specify.

For Integrity servers objects, the Elf header, the section header table and the note section are always analyzed, no matter which qualifiers you specify.

Qualifiers

/DISASSEMBLE (Integrity servers only)

Positional qualifier.

Displays all sections of type SHT_PROGBITS where the executable flag is set (SHDR$M_SHF_EXECINSTR in the section header). The section data will be displayed as machine instructions with symbolization of labels, branch targets, and so on. All local and global symbols from the symbol table are used for symbolization. The output is similar to compiler generated machine code listings.

Note

This qualifier is accepted only for objects. Integrity servers images contain only global symbols, if any at all. In addition, output produced with this qualifier differs from output produced by **ANALYZE/OBJECT/SECTIONS=CODE**, which provides machine code output for the same sections, although without symbolization.

/DBG (Alpha only)

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 (Alpha only)

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 *VSI OpenVMS Linker Utility Manual* [https://docs.vmssoftware.com/vsi-openvms-linker-utility-manual/] for more information.

/FLAGVALUES= (*keyword*[,...]) (Integrity servers only)

Several fields in an ELF module represent bit flags. Where possible, these bit-flag values are examined and displayed individually. By default, only the flag values that are set to 1 (ON) are displayed. The keywords are as follows:

Keyword	Description
ON	Displays all flags whose value is 1.
OFF	Displays all flags whose value is 0.
ALL	Displays all flag values. The keywords ON and OFF, in contrast, indicate the value of each specific flag bit.

/GSD (Alpha only)

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 (Alpha only)

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 (Alpha only)

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.

The asterisk (*) and the percent sign (%) wildcard characters are not allowed in the file specification.

/PAGE_BREAK=*keyword* (Integrity servers only)

Specifies if and where page breaks (form feeds) are inserted in the report file. This qualifier is only useful if **/OUTPUT** is used to write a report file. It is ignored if **/INTERACTIVE** is used to specify an interactive analysis.

The keywords are as follows:

Keyword	Description
NONE	Creates a report without any page break.
PRINTABLE_REPORT	Creates a printable report with page breaks as in listing files. The number of lines per page is the default number

Keyword	Description
	of lines on a printer page. This is the default behavior for ANALYZE_OBJECT when no qualifier is not specified.
SEPARATE_INFORMATION	Inserts a page break between different section information.

/SECTIONS [=(*keyword*[,...])] (Integrity servers only)

Selects individual program sections or section types to display.

Note

This qualifier and its keywords can only be used to form an inclusion list of sections to be displayed. This qualifier is not negatable and cannot be used to form an exclusion list. If no values are specified, the default keyword is HEADERS.

The keywords are as follows:

Keyword	Description
ALL	Displays a detailed analysis of every section in the module. Note that this keyword can generate a large amount of output.
CODE	Displays all sections of type SHT_PROGBITS where the executable flag is set (SHDR$M_SHF_EXECINSTR in the section header). The section data will be displayed as machine instructions.
DEBUG [=(*suffix*[,...])]	Analyzes and displays sections consisting of debug formatted debug information. In addition, you can use a list of debug section name suffixes to selectively format DEBUG information. The debug section names, which appear as ".debug_suffix", can be viewed in the summary table. The suffix can be specified as follows: ● ABBREV – Formats DEBUG abbreviations ● ARANGES – Formats DEBUG address lookup tables ● FRAME – Formats DEBUG frame descriptors for unwinding ● INFO – Formats DEBUG symbols ● LINE – Formats DEBUG source line info ● PUBNAMES – Formats DEBUG name lookup tables ● PUBTYPES – Formats DEBUG type lookup tables
EXTENSIONS	Analyzes and displays sections of type SHT_IA64_EXT. The data is displayed in hexadecimal format.
GROUP	Analyzes and displays sections of type SHT_GROUP. Sections of this type consist of a list of the section numbers of sections belonging to that group.

Keyword	Description
HEADERS	The default keyword. Displays the ELF header and the section header details.
LINKAGES	Analyzes and displays sections of type SHT_VMS_LINKAGES. The data is displayed as a list of linkage descriptors.
NOBITS	Analyzes and displays sections of type SHT_NOBITS. There is no module data associated with sections of this type.
NOTE	Analyzes and displays sections of type SHT_NOTE. The data for this section is displayed as a list of formatted OpenVMS note entries.
NULL	Displays all sections of type PT_NULL. No data will be displayed for segments of this type.
NUMBERS=(*number* [,...])	Displays individual sections, as follows: ● The selected sections will have a detailed display of their header and their contents. An informational message is displayed for section numbers that do not exist in the module. ● One or more numeric values may be specified. ● Section numbers may be specified in decimal, octal (using the %O prefix), or hexadecimal (using the %X prefix).
PROGBITS	Displays all sections of type SHT_PROGBITS, except unwind sections. Formatting for the sections of type SHT_PROGBITS depends on the EXECINSTR flag (SHDR$M_SHF_EXECINSTR) in its section header. If this bit is set, the section data will be displayed as machine instructions. Otherwise, it will be displayed as hexadecimal data. Unwind sections will be displayed if **/SECTIONS=UNWIND** is specified.
RELOCATIONS	Analyzes and displays sections of type SHT_RELA. The data for this section is displayed as table of relocation entries.
STRTAB	Analyzes and displays sections of type SHT_STRTAB. The data for this section is displayed as a string table.
SYMTAB	Displays sections of type SHT_SYMTAB. The data for this section is displayed as a symbol table.
TRACE [=(*suffix*[,...])]	Analyzes and displays sections consisting of traceback information. In addition, you can use a list of trace section name suffixes to selectively format TRACE information. The trace section names, which appear as ".trace_suffix", can be viewed in the summary table. The suffix can be specified as shown below. In addition, because there is one common debug and traceback

Keyword	Description
	section, ".debug_line", the suffix "line" can be specified as shown below as well: • ABBREV – Formats TRACE abbreviations • ARANGES – Formats TRACE address lookup tables • INFO – Formats TRACE symbols • LINE – Formats TRACE source line info
UNWIND	Analyzes and displays sections of type SHT_IA64_UNWIND. Each section of this type has an associated Unwind Information section of type SHT_PROGBITS. This associated section is also displayed.

/SELECT=(*keyword*[,...])

Allows for the collection of specific object file information and displays the selected keyword items in the order specified.

Note

The **/SELECT** qualifier can be used on object and image files. The same keywords are valid selections. However, some information can not be in an object, such as the link date and time. Therefore, for some keywords the Analyze utility returns "Unknown". In the following table, only the keywords (which are useful for object files) and their return values are listed.

Analyze creates DCL symbols for all selectable information with the **/SELECT** qualifier. The symbol names consist of the prefix ANALYZE$ and a descriptive name of the information they hold. The symbol value is the selected information, usually printed to SYS$OUTPUT. Effectively, all of the printed information is duplicated in the symbols. For unselected information, the corresponding symbols will contain the null string.

The keywords are as follows:

Keyword	Description
ARCHITECTURE	Writes the architecture information into the DCL symbol ANALYZE$ARCHITECTURE. Returns "OpenVMS IA64" if the file is an OpenVMS Integrity servers object file. an OpenVMS Alpha object file returns "OpenVMS Alpha" if the file is an OpenVMS Alpha object file.
FILE_TYPE	Writes file type information into the DCL symbol ANALYZE$FILE_TYPE. Returns "Object" if the file is an OpenVMS Integrity servers or Alpha object file.

/TBT (Alpha only)

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 (Alpha only)

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`. 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 (Alpha only)`

   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`.

3. ```
   $ ANALYZE/OBJECT/SELECT=(ARCH,FILE) *.OBJ
   DISK:[DIRECTORY]ALPHA.OBJ;1
      OpenVMS ALPHA
      Object
   ```

   This example displays the information requested about the object files `ALPHA.OBJ`.

ANALYZE/PROCESS_DUMP

ANALYZE/PROCESS_DUMP — Invokes the OpenVMS 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).

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.
   ```

Note

Requires read (R) access to the dump file.

The OpenVMS Debugger is invoked automatically. For a complete description of the debugger, including information about the DEBUG command, see the *VSI OpenVMS Debugger Manual* [https://docs.vmssoftware.com/vsi-openvms-debugger-manual/]. 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. On Alpha systems, you can use the **DUMP/PROCESS** command.

For OpenVMS Alpha Systems

This section applies to Alpha systems running Version 7.2 or before.

Note

VSI strongly recommends that you analyze a process dump on the system where the dump was generated. It is highly unlikely that you can analyze a dump successfully if you move the dump file to a different system.

Different configurations can cause the process executing the **ANALYZE/PROCESS_DUMP** command to fail to load the dumped image successfully. For example, if the systems have different versions of the operating system, the analysis might work, but it is not guaranteed.

Other restrictions include the configuration of the control regions in P1 space, the process running at the time of the dump, and the process performing the **ANALYZE/PROCESS_DUMP** command. The location of the base of the user stack for each process, which depends on the size of allocated space, determines whether the processes are compatible. The size of allocated space for the process analyzing the dump must be less than the size of allocated space for the process that created the dump. If you are analyzing the dump on a different system, but with the same version of the operating system, you can decrease the size of allocated space by modifying one or more of the system parameters that affect the size of allocated space.

You can modify the system parameter IMGIOCNT dynamically. Other parameters to adjust allocated space require a reboot of the system.

On Alpha systems running version 7.2 or before, the system parameter IMGREG_PAGES is likely to cause a problem with allocated size. When a dump comes from a system without DECwindows and is examined on a system with DECwindows, a P1 message is displayed. DECwindows requires IMGREG_PAGES to be at least 2000 pages, which means that the value is too large by 1200 to 1400 pages.

Also, in some cases, the OpenVMS Debugger is incapable of analyzing the dumped image. For example, when the dumped image's PC is set to an invalid address or when the dumped image's stack is corrupted by a bad process descriptor, you must use the Delta Debugger (DELTA) to analyze the dump. To use DELTA as the debugger, you must install the SYS$LIBRARY:DELTA image by invoking the Install utility. For complete information on the Install utility, see the relevant section in the *VSI OpenVMS System Management Utilities Reference Manual, Volume 1: A-L* [https://docs.vmssoftware.com/vsi-openvms-system-management-utilities-reference-manual-volume-i-a-l/#INSTALL_CH].

This section applies to OpenVMS Alpha systems running Version 7.3 or greater.

You can now analyze a dump file on a system other than where the dump was generated. However, if the base image link date and time are not the same, you will need to also copy the file, SYS$BASE_IMAGE.EXE from the generating system, and point to it using the logical, SDA$READ_DIR. For example:

```
$ COPY other_node::SYS$LOADABLE_IMAGES:SYS$BASE_IMAGE.EXE my_disk$:
[my_dir]
$ DEFINE/USER SDA$READ_DIR my_disk$:[my_dir],SYS$SYSROOT:[SYS$LDR], -
  SYS$SYSROOT:[SYSLIB]
$ ANALYZE/PROCESS_DUMP mycrash.dmp
```

If you are analyzing a threaded process dump on a system other than the system on which it was generated, you may also need to copy and point to PTHREAD$RTL and PTHREAD$DBGSHR (DECthread debug assistant) on the generating system. For example:

```
$ COPY other_node::SYS$LOADABLE_IMAGES:SYS$BASE_IMAGE.EXE my_disk$:
[my_dir]
$ COPY other_node::SYS$SHARE:PTHREAD$RTL.EXE my_disk$:[my_dir]
$ COPY other_node::SYS$SHARE:PTHREAD$DBGSHR.EXE my_disk$:[my_dir]
$ DEFINE/USER SDA$READ_DIR my_disk$:[my_dir],SYS$SYSROOT:[SYS$LDR], -
  SYS$SYSROOT:[SYSLIB]
$ DEFINE/USER PTHREAD$RTL my_disk$:[my_dir]PTHREAD$RTL.EXE
$ DEFINE/USER PTHREAD$DBGSHR my_disk$:[my_dir]PTHREAD$DBGSHR.EXE
$ ANALYZE/PROCESS_DUMP mycrash.dmp
```

If you are unable to analyze a process dump with the debugger, then you should attempt to use the System Dump Analyzer (SDA) utility. See the **ANALYZE/CRASH** command in online help for more information. For example:

```
$ ANALYZE/CRASH mycrash.dmp

OpenVMS (TM) Alpha system dump analyzer
...analyzing a compressed process dump...

Dump taken on 19-OCT-1999 12:03:40.95
SDA> ..
.
.
.
```

Qualifiers

/FULL

On Alpha systems, shows the information that is displayed by the following debugger commands: **SHOW IMAGE**, **SHOW THREAD/ALL**, and **SHOW CALL**.

/IMAGE_PATH[=*directory-spec*] *dump-file*
/NOIMAGE_PATH

On Alpha systems, specifies the search path the debugger is to use to find the debugger symbol table (DST) file. As in prior debuggers, the debugger builds an image list from the saved process image list. When you set an image (the main image is automatically set), the debugger attempts to open that image in order to find the DST file.

If you include the **/IMAGE_PATH=*directory-spec*** qualifier, the debugger searches for the DST file in the specified directory. The debugger first tries to translate *directory-spec* as the

logical name of a directory search list. If that fails, the debugger interprets *directory-spec* as a directory specification, and searches that directory for matching .DSF or .EXE files. A .DSF file takes precedence over an .EXE file. The name of the .DSF or .EXE file must match the image.

If you do not include the **/IMAGE_PATH=*directory-spec*** qualifier, the debugger looks for the DST file first in the directory that contains the dump file. If that fails, the debugger searches directory SYS$SHARE and then directory SYS$MESSAGE. If the debugger fails to find a DST file for an image, the symbolic information available to the debugger is limited to global and universal symbol names.

Version 7.3 and later debuggers check for dump file image specification and DST file link date-time mismatches and issue a warning if one is discovered.

The *dump-file* parameter is the name of the process dump file to be analyzed. Note that the process dump file type must be .DMP and the DST file type must be either .DSF or .EXE.

Restrictions

You cannot use a logical to redirect the search for an image and use the **/IMAGE_PATH** qualifier at the same time. If you use the **/IMAGE_PATH** qualifier, then all images that are not in their original locations must be found through that path. Individual image logicals (for example, the "SH" in "DEFINE SH SYS$LOGIN:SH.EXE") are not processed.

Additionally, you cannot input a directory search path directly to the **/IMAGE_PATH** qualifier, as it does not process a directory list separated by commas; however, you can specify a logical that translates into a directory search path.

Examples

```
$ ANALYZE/PROCESS/FULL WECRASH.DMP

           OpenVMS Alpha Debug64 Version X7.3-010
%SYSTEM-F-IMGDMP, dynamic image dump signal at PC=001D0F8CB280099C, PS=001D0028
break on unhandled exception preceding WECRASHth_run%LINE 26412 in THREAD 8
%DEBUG-W-UNAOPNSRC, unable to open source file DSKD$:[IMGDMP]WECRASH.C;11
-RMS-F-DEV, error in device name or inappropriate device type for operation
 26412: Source line not available

 image name            set    base address           end address
 CMA$TIS_SHR           no     000000007B8CA000       000000007B8D7FFF
    CODE0                     FFFFFFFF80500000       FFFFFFFF805033FF
    DATA1                     000000007B8CA000       000000007B8CB3FF
    DATA2                     000000007B8CC000       000000007B8D13FF
    DATA3                     000000007B8D2000       000000007B8D21FF
    DATA4                     000000007B8D4000       000000007B8D41FF
    DATA5                     000000007B8D6000       000000007B8D63FF
 DECC$SHR              no     000000007BE7A000       000000007BF0DFFF
    CODE0                     FFFFFFFF8055C000       FFFFFFFF806C9DFF
    DATA1                     000000007BE7A000       000000007BEACFFF
    DATA2                     000000007BEBA000       000000007BEC2DFF
    DATA3                     000000007BECA000       000000007BED77FF
    DATA4                     000000007BEDA000       000000007BEDA9FF
    DATA5                     000000007BEEA000       000000007BEEA1FF
    DATA6                     000000007BEFA000       000000007BEFE7FF
    DATA7                     000000007BF0A000       000000007BF0D1FF
 DPML$SHR              no     000000007BB92000       000000007BBD1FFF
    CODE0                     FFFFFFFF80504000       FFFFFFFF8055B5FF
    DATA1                     000000007BB92000       000000007BBAC1FF
    DATA2                     000000007BBAE000       000000007BBBDBFF
    DATA3                     000000007BBBE000       000000007BBBE1FF
```

```
    DATA4                    000000007BBC0000       000000007BBCC9FF
    DATA5                    000000007BBCE000       000000007BBCE3FF
    DATA6                    000000007BBD0000       000000007BBD07FF
 LIBOTS                 no   000000007B5AA000       000000007B5B1FFF
    DATA1                    000000007B5AA000       000000007B5AC5FF
    DATA2                    000000007B5AE000       000000007B5AFBFF
    DATA3                    000000007B5B0000       000000007B5B01FF
 LIBRTL                 no   000000007B558000       000000007B5A9FFF
    CODE0                    FFFFFFFF8041C000       FFFFFFFF804BD7FF
    DATA1                    000000007B558000       000000007B5669FF
    DATA2                    000000007B568000       000000007B5697FF
    DATA3                    000000007B578000       000000007B5845FF
    DATA4                    000000007B588000       000000007B5881FF
    DATA5                    000000007B598000       000000007B59A5FF
    DATA6                    000000007B5A8000       000000007B5A99FF
 PTHREAD$RTL            no   000000007BBD2000       000000007BC27FFF
    DATA0                    000000007BBD2000       000000007BBDA1FF
    DATA1                    000000007BBDC000       000000007BBDF3FF
    DATA2                    000000007BBE0000       000000007BBE2FFF
    DATA3                    000000007BBE4000       000000007BC1E1FF
    DATA4                    000000007BC20000       000000007BC20BFF
    DATA5                    000000007BC22000       000000007BC247FF
    DATA6                    000000007BC26000       000000007BC275FF
*WECRASH                yes  0000000000010000       00000000000403FF

 total images: 7

  Thread Name                    State           Substate  Policy       Pri
  ------ ---------------------- --------------- --------- ------------ ---
       1 default thread          blocked         join    2 SCHED_OTHER  11
       2 thread 0: counting      ready VP 0                SCHED_OTHER  11
       3 thread 1: dumping       ready VP 0                SCHED_OTHER  11
       4 thread 2                blocked         delay     SCHED_OTHER  11
       5 thread 3                blocked         delay     SCHED_OTHER  11
       6 thread 4                blocked         delay     SCHED_OTHER  11
       7 thread 5: counting      ready VP 0                SCHED_OTHER  11
       8 thread 6: dumping       running                   SCHED_OTHER  11
       9 thread 7                blocked         delay     SCHED_OTHER  11
      10 thread 8                blocked         delay     SCHED_OTHER  11
      11 thread 9                blocked         delay     SCHED_OTHER  11

 module name    routine name  line     rel PC           abs PC
*WECRASH        th_run       26411 0000000000000244 0000000000030244
 SHARE$PTHREAD$RTL_DATA0           000000000001F15C 000000007BC0315C
 SHARE$PTHREAD$RTL_DATA0           000000000000F494 000000007BBF3494
                                   0000000000000000 0000000000000000
----- the above looks like a null frame in the same scope as the frame
below SHARE$PTHREAD$RTL_DATA0                        ?              ?

DBG>
DBG> set source/latest sys$disk:[]
DBG> examine/source .pc-4
module WECRASH
 26411:         lib$signal(SS$_IMGDMP);
DBG>
```

This example shows the output of the **ANALYZE/PROCESS** command on a multithreaded process dump, using the **/FULL** qualifier on an Alpha system.

ANALYZE/RMS_FILE

ANALYZE/RMS_FILE — Invokes the Analyze/RMS_File utility, which is used to inspect and analyze the internal structure of an OpenVMS RMS file. The **/RMS_FILE** qualifier is required. For more

information about the Analyze/RMS_File utility, see the relevant section in the *VSI OpenVMS Record Management Utilities Reference Manual* [https://docs.vmssoftware.com/vsi-openvms-record-management-utilities-reference-manual/#ANALYZE_RMS_FILE] or online help.

Format

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

ANALYZE/SSLOG (Alpha/Integrity servers Only)

ANALYZE/SSLOG (Alpha/Integrity servers Only) — Analyzes the `SSLOG.DAT` file, which contains system service logging data. The **/SSLOG** qualifier is required. For more information, see the online help for **ANALYZE/SSLOG** or read the chapter about system service logging in the *VSI OpenVMS System Analysis Tools Manual* [https://docs.vmssoftware.com/vsi-openvms-system-analysis-tools-manual/#CHAPT14].

Format

ANALYZE/SSLOG [*qualifiers*] [*filespec*]

ANALYZE/SYSTEM

ANALYZE/SYSTEM — Invokes the System Dump Analyzer utility, which analyzes a running system. The **/SYSTEM** qualifier is required. For more information about the System Dump Analyzer utility on Alpha and Integrity server systems, see the relevant section in the *VSI OpenVMS System Analysis Tools Manual* [https://docs.vmssoftware.com/vsi-openvms-system-analysis-tools-manual/#PART1] or online help.

Format

ANALYZE/SYSTEM

APPEND

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 each file specification with either a comma (,) or a plus sign (+).

The asterisk (*) and the percent sign (%) wildcard characters 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 OpenVMS File Applications *[https://docs.vmssoftware.com/guide-to-openvms-file-applications/]* [https://docs.vmssoftware.com/guide-to-openvms-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 absolute time, as a combination of absolute and delta times, or as one of the following keywords: BOOT, LOGIN, 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 relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT] or the online help topic Date.

/BLOCK_SIZE=*n*

Overrides the default block size (124) used by **COPY**. You can specify a value in the range of 1 through 127.

/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 relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#UIC_FORMAT_DIR].

/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 **Return**. 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 the *ownership* parameter as system (S), owner (O), group (G), or world (W).

- Specify 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. This qualifier is relevant only with the **/NEW_VERSION** qualifier.

For more information on specifying protection codes, see the relevant section in the *VSI OpenVMS Guide to System Security* [https://docs.vmssoftware.com/vsi-openvms-guide-to-system-security/#PROTECT_CODE_DETAILS].

/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 on or after the specified time. You can specify time as absolute time, as a combination of absolute and delta times, or as one of the following keywords: BOOT, JOB_LOGIN, LOGIN, 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 relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT] or the online help topic Date.

/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"BILL_BESTON YANKEE"::DEMO1.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

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 sets of 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. A logical name can have a maximum of 128 equivalence names.

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 relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#LOGNAME17].

***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 sets of 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, the name can have only 1 to 31 alphanumeric characters (including the dollar sign and underscore). If the logical name being entered into the process or system directory translates to a logical name table name, any alphabetic characters in the name should all be uppercase. 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 sets of 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.

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**, **/SUPERVISOR_MODE**, or **/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 creating and using logical names, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#LOGNAMES_CH].

Qualifiers

/CLUSTER_SYSTEM

You must be signed in to the SYSTEM account or have SYSNAM (system logical name) or SYSPRV (system) privilege to use this qualifier.

Assigns a clusterwide logical name in the LNM$SYSCLUSTER table.

/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:

Keyword	Description
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:

Keyword	Description
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. Also, user-mode logical names are automatically deleted when invoking and exiting a command procedure.

Examples

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

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

2.
```
$ ASSIGN/USER_MODE $DISK1:[FODDY.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:[HEROLD]  ED
$ PRINT  ED:TEST.DAT
Job 274 entered on queue SYS$PRINT
```

The **ASSIGN** command in this example associates the logical name `ED` with the directory name `[HEROLD]` on the disk `XXX1`. Subsequent references to the logical name `ED` result in the correspondence between the logical name `ED` and the disk and directory specified. The **PRINT** command queues a copy of the file XXX1:[HEROLD]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  DKA1:  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 DKA1, 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  DKA0:  SYSFILES
$ SHOW LOGICAL  SYSFILES
  "SYSFILES" = "DKA0:" (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.

10.
```
$ 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

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

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.

Note

Requires manage (M) access to both queues.

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 re-queued for processing on the target queue. If the **STOP/QUEUE/REQUEUE** command fails to re-queue 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 re-queues 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

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.

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.

Note

Requires manage (M) access to both queues.

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

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

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.

Note

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

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 in the *VSI OpenVMS DCL Dictionary: N–Z* [https://docs.vmssoftware.com/vsi-openvms-dcl-dictionary-n-z/#BRASS_134] 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

BACKUP — Invokes the Backup utility (BACKUP) to perform the backup operations.

Format

BACKUP*input-specifier output-specifier*

Description

You can perform 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.

You cannot invoke **BACKUP** to back up a system disk. A system disk must be bootstrapped to run.

For more information about **BACKUP** and backing up the system disk, see the relevant sections in the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/#_6017BACKUP] and the *VSI OpenVMS System Management Utilities Reference Manual, Volume 1: A-L* [https://docs.vmssoftware.com/vsi-openvms-system-management-utilities-reference-manual-volume-i-a-l/#BKU_U] or online help.

CALL

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
$A: SUBROUTINE
$ 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 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 spaces.

Setting bit 3 of DCL_CTLFLAGS to 1, specifies from one to sixteen optional parameters to pass to the command procedure. Use 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 sixteen. The symbols are local to the specified command procedure. Separate each parameter with one or more spaces. If you clear the bit 3 of DCL_CTLFLAGS, the default parameters are set (that is, (P1, P2, … P8)).

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 sets of 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. When bit 3 of DCL_CTLFLAGS is set to 1, you can use the **`CALL`** command to transfer control to a subroutine, a new procedure level is created and the symbols P1 to P16 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 it her 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 bit 3 of DCL_CTLFLAGS is set to 1, **CALL** command allows you to define the parameters up to P16. 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. The asterisk (*) and the percent sign (%) wildcard characters are not 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.

Examples

```
$
$! 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:  !Exit this command procedure file
$ 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

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

Format

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

Parameters

node-name::

The name of the node on which the specified process is running.

You cannot specify a node name on a different OpenVMS Cluster system 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.

Note

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

---

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 sub process 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.

CHECKSUM

CHECKSUM — Invokes a utility to calculate one or more checksums for OpenVMS files. The result, or checksum, is available in the DCL symbol CHECKSUM$CHECKSUM.

Format

CHECKSUM *filespec*

Parameter

filespec

Specifies the name of an existing file to be checksummed. The asterisk (*) and percent sign (%) wildcard characters are allowed in the file specification.

Description

The CHECKSUM utility calculates file, image, or object checksums for an OpenVMS file. For a file checksum the algorithm used determines if the internal record structure of the file is followed or not. For an image or object checksum, the utility always follows the image or object structure.

The **/FILE**, **/IMAGE**, and **/OBJECT** qualifiers determine which kind of checksum is calculated. They imply a default file type (.DAT, .EXE or .OBJ) and determine the amount of information displayed. The default, **/FILE**, results in an XOR file checksum, according to the file's record structure. It implies a default file type .DAT and determines that no information is output to SYS$OUTPUT.

For file checksums, you can specify which algorithm **CHECKSUM** will use to perform calculations. By default, the XOR record-based algorithm is used. Optionally, you can select the CRC, MD5, SHA1, or SHA256 algorithm.

The CRC algorithm is the same as the algorithm used for ELF-64 files and is used by popular compression tools like PKZIP. That is, a file checksum in a ZIP file can be compared with the file checksum obtained by the CHECKSUM utility. The MD5 algorithm is the MD5 digest, which can be obtained using public domain tools such as MD5.EXE and md5sum.

Image checksums differ between the Alpha platforms and the Integrity servers platform. Object checksums are only available for the Integrity servers platform. With the platform qualifiers, such as **/ALPHA**, **/I64**, or **/VAX**, non-native images or objects can be checksummed.

For all ELF-64 image and object checksums, **CHECKSUM** uses a CRC-32 algorithm. The CRC, known as AUTODIN II, Ethernet, or FDDI CRC, is documented as part of the VAX CRC instructions. The image or object checksum follows the ELF-64 data structures that are used for OpenVMS Integrity servers object and image files. For these checksums, only the invariant data is used for the calculation. Variant data, such as timestamps and versions, are excluded from the checksum calculation in order to compare results from different compile and link operations.

For Alpha and VAX images, **CHECKSUM** uses an XOR algorithm. The image checksum follows the Alpha and VAX image structure and only uses invariant data for the calculation. Variant data, such as timestamps are excluded in order to compare results from different link operations. Note that on Alpha and VAX systems, object files cannot be checksummed based on object invariant data.

Qualifiers

/ALGORITHM=*option*
/ALGORITHM=XOR (default)

Selects the algorithm used for file checksums. The default is the XOR algorithm for data within records. Options include:

- CRC – A CRC-32 algorithm for all bytes within the file (possible record structures are ignored); this algorithm is also known as AUTODIN II, Ethernet, or FDDI CRC.

- MD5 – The MD5 digest, as published by Ronald L. Rivest (RFC 1321), for all bytes within the file (possible record structures are ignored).
- SHA1 – A Secure Hash Algorithm for all bytes within the file (possible record structures are ignored). A 160-bit (20-byte) hash function designed by the National Security Agency (NSA).
- SHA256 – A Secure Hash Algorithm for all bytes within the file (possible record structures are ignored). A 256-bit (32-byte) hash function designed by the National Security Agency (NSA).
- XOR – An XOR algorithm for all data, according to the record structure of the file.

/ALPHA

Calculates an Alpha-type checksum and is only useful with the **/IMAGE** qualifier on Integrity server systems (that is, it checksums Alpha images on Integrity server systems). It is set by default on Alpha platforms.

/FILE (default)

Calculates a file checksum.

By default, the XOR algorithm (**/ALGORITHM=XOR**) is used for the checksum. The **/FILE** qualifier also implies a default file type of .DAT. By default, unsigned decimal checksum value is saved in the DCL symbol CHECKSUM$CHECKSUM and not output to the screen. By specifying **/SHOW=DATA**, the full file name of the specified input file is output in addition to the file checksum, an unsigned decimal value.

The **/ALPHA**, **/I64**, or **/VAX** platform qualifiers do not influence the file checksum result. However, **/ALPHA** and **/VAX** prohibit the **/SHOW** qualifier because these qualifiers were not available on the original Checksum utility for Alpha systems.

/I64

Calculates an I64-type checksum and is only useful on Alpha systems with **/IMAGE** or **/OBJECT** (that is, it checksums Integrity servers images or objects on Alpha systems). The **/I64** qualifier is set by default on Integrity servers platforms.

/IMAGE

Calculates a checksum of all image bytes. The image structure is followed to include only the image bytes into the checksum. Invariant data, such as the linker version and the link date, are omitted.

For Integrity servers images (that is, Integrity servers formatted files), a CRC checksum is calculated and additional information is output to SYS$OUTPUT, including the following:

- The resulting full file name and checksums for the image segments
- The header checksums and the overall image checksum

The output values are shown in hexadecimal notation. The DCL symbol, CHECKSUM$CHECKSUM, shows the result in hexadecimal notation.

For Alpha and VAX images, an XOR checksum is calculated and additional information is output to SYS$OUTPUT:

- The resulting full file name and checksums for the image sections
- The header checksum and the overall image checksum

The output checksum values are in hexadecimal notation. However, the result in the DCL symbol CHECKSUM$CHECKSUM is in unsigned decimal notation.

Note

For Alpha and VAX images, the unsigned decimal notation of the checksum value in the DCL symbol CHECKSUM$CHECKSUM retains compatibility with the previous checksum tool.

The **/IMAGE** qualifier implies the default file type of .EXE. For Integrity servers images, this qualifier also implies the default keyword values HEADERS and SEGMENTS for the **/SHOW** qualifier.

/OBJECT

Calculates a CRC checksum of all Integrity servers object bytes.

The **/OBJECT** qualifier follows the ELF-64 object structure to include only the object bytes into the checksum. Invariant data, as the language processor version and the generation date, are omitted.

Additional information is output to SYS$OUTPUT, including the following:

- The resulting full file name of the specified input file
- The checksums for the object sections, headers, and the overall object checksum

The output checksum values are in hexadecimal notation. The result provided in the DCL symbol, CHECKSUM$CHECKSUM, is in hexadecimal notation.

The **/OBJECT** qualifier implies the default file type of .OBJ. This qualifier also implies the default keyword values HEADERS and SECTIONS for the **/SHOW** qualifier.

On Alpha platforms, it is only applicable with the **/I64** qualifier.

/OUTPUT[=*filespec*]
/NOOUTPUT

The **/OUTPUT** qualifier controls where the output of the command is sent. The **/NOOUTPUT** qualifier suppresses output.

If you specify **/OUTPUT** and a file specification (**/OUTPUT=filespec**), the output is sent to the specified file, rather than to the current output device, SYS$OUTPUT. If you do not enter the qualifier, or if you enter the **/OUTPUT** qualifier without a file specification, the output is sent to SYS$OUTPUT.

Using the **/OUTPUT** qualifier does not affect the result (that is, the DCL symbol CHECKSUM$CHECKSUM).

/SHOW=(*option*[,...])

Controls which checksum and additional information is output to the device.

Options for this qualifier are as follows:

- ALL – Sets all of the applicable options, with the following restrictions:
  - For file checksums, only the DATA keyword is allowed.
  - For image checksums, all keywords are allowed.
  - For object checksums, the SEGMENT keyword is not allowed.
- DATA – Outputs the full file name and the file checksum. For compatibility, this option is available for **/FILE**.
- EXCLUDED – Formats the data excluded from the image or object checksums.
- HEADERS – Output checksums of all Integrity servers headers. This option is set by default for **/IMAGE** and **/OBJECT**.
- SECTIONS – Output checksums of all ELF-64 sections. This option is set by default for **/OBJECT**.
- SEGMENTS – Output checksums of all ELF-64 program segments. This option is set by default for **/IMAGE**.

/VAX

Calculates a VAX-type checksum and is only useful on Integrity servers or Alpha systems with **/IMAGE** to checksum VAX images on non-VAX systems.

Examples

The **CHECKSUM/IMAGE** command results in different output for Integrity servers and Alpha platforms. Because there are different image structures, the names for the checksums differ:

- The checksum for Alpha outputs the section number as BLISS constant: %D'1' whereas the Integrity servers checksum outputs decimal numbers.
- The checksum for Alpha outputs the checksums as BLISS constant: %X'6C5404CB' whereas the Integrity servers checksum outputs DCL-style hexadecimal numbers.
- The DCL symbol on Alpha is an unsigned decimal value, whereas the DCL symbol for Integrity servers is a hexadecimal value.

On Alpha systems:

```
$ CHECKSUM/IMAGE HELLO.EXE
file DISK$USER:[JOE]HELLO.EXE;10
image section %D'1' checksum is %X'6C5404CB'
image section %D'2' checksum is %X'E29D6A3A'
image section %D'3' checksum is %X'114B0786'
image header checksum is %X'00000204'
checksum of all image sections is %X'9F826977'

$ SHOW SYMBOL CHECKSUM$CHECKSUM
CHECKSUM$CHECKSUM = "2676124023"
```

On Integrity server systems:

```
$ CHECKSUM/IMAGE FOOBAR.EXE
File DISK$USER:[JOE]FOOBAR.EXE;3
Checksum program segment 0: %X18E293D7
Checksum program segment 1: %XEFBCE000
Checksum program segment 2: %XA6D02DD5
Checksum program segment 3: %X30130E3E
Checksum dynamic segment %X0F704080
Elf header checksum: %X7A6AC80F
Elf program header checksum: %XBF6B41D8
Elf section header checksum: %X6C770CF6
Elf (object/image) checksum: %X2EEE7726

$ SHOW SYMBOL CHECKSUM$CHECKSUM
CHECKSUM$CHECKSUM = "2EEE7726"
```

CLOSE

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

/DISPOSITION=*option*

Specifies what action to take when the file is closed. The options are:

Option	Description
DELETE	Delete the file.
KEEP (default)	Keep the file.
PRINT	Print the file.
SUBMIT	Submit the file.

/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 betaken.

/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 **SHOWLOGICAL/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

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

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.

Note

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.

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 Manager utility (SYSMAN) on OpenVMS Alpha systems.

If virtual terminals are allowed on your system, use the **SETTERMINAL/DISCONNECT/PERMANENT** command to enable the virtual terminal characteristic for a particular physical terminal. When you enable this characteristic, a virtual terminal is 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.

For new virtual terminals, you must first set the TT2$V_DISCONNECT bit in the TTY_DEFCHAR2 system parameter and reboot the system. This is done by creating the virtual device VTA0: using the ttdriver. For example, on Alpha:

```
$ RUN SYS$SYSTEM:SYSMAN
SYSMAN> IO CONNECT/NOADAPTER/DRIVER=SYS$LOADABLE_IMAGES:SYS$TTDRIVER VTA0:
```

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/FULL
OpenVMS User Processes at 21-JUL-2009 14:11:56.91
Total number of users = 51, number of processes = 158
Username Node Process Name PID Terminal
KIDDER BUKETT KIDDER 29A0015E FTA3:
KIDDER BUKETT _FTA4: 29A0015F FTA4:
KIDDER RACEY1 KIDDER 05800062 FTA5:
KIDDER RACEY1 DECW$MWM 0580005D MBA44: Disconnected
KIDDER RACEY1 DECW$SESSION 05800059
KIDDER RACEY1 VUE$KIDDER_2 0580005E (subprocess of 05800059)
KIDDER RACEY1 VUE$KIDDER_3 0580005F MBA51: Disconnected
KIDDER RACEY1 VUE$KIDDER_4 05800060 MBA53: Disconnected
SMITH BUKETT SMITH 29A002C1 FTA7:
SMITH BUKETT SMITH_1 29A006C2 (subprocess of 29A002C1)
SMITH BUKETT SMITH_2 29A00244 (subprocess of 29A002C1)
SMITH HAMLET SMITH 24800126 FTA6:
SMITH HAMLET DECW$BANNER 24800155 (subprocess of 24800126)
SMITH HAMLET DECW$MWM 2480011F MBA170: Disconnected
SMITH HAMLET DECW$SESSION 2480011D FTA5:
  .
```

```
   .
   .
$ CONNECT VTA273
SMITH logged out at 22-DEC-2001 14:12:04.53
$
```

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/FULL** 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 SMITH was at interactive level when the connection was broken.

CONTINUE

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 *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-users-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.

Examples

1.
```
$ RUN MYPROGRAM_A
Ctrl/Y
$ SHOW TIME
  14-DEC-2001 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.

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

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 more information about the Convert utility, see the relevant section in the *VSI OpenVMS Record Management Utilities Reference Manual* [https://docs.vmssoftware.com/vsi-openvms-record-management-utilities-reference-manual/#CONVERT_FILE] or online help.

Format

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

CONVERT/DOCUMENT

CONVERT/DOCUMENT — Converts a CDA supported revisable input file to another revisable or final form output file.

Format

CONVERT/DOCUMENT*input-filespec output-filespec*

Parameters

input-filespec

Specifies the name of the input file to be converted. The default file type is .DDIF.

output-filespec

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

Description

The **CONVERT/DOCUMENT** command lets you convert documents from one format to another.

Note

You can use this command only if DECwindows Motif for OpenVMS is installed on your system.

You specify the name and format of the input file (a file whose format is incompatible with the application that needs to read the file) and the output file (the file to be created in a new format).

You can convert a file from one format to another if an input converter exists for the input file format and an output converter exists for the output file format. The default input and output file format is DDIF (DIGITAL Document Interchange Format). DDIF is a standard format for the storage and interchange of compound documents, which can include text, graphics, and images.

DDIF input and output converters, in addition to several other converters, are installed with the CDA Base Services for DECwindows Motif for OpenVMS. Some of the converters support processing options, which ensure minimal changes when your input file is converted to a different output file format. Create an options file with the processing options you need before specifying the **CONVERT/DOCUMENT** command with the **/OPTIONS** qualifier.

Every converter supports a message log option, which is a file name you specify and to which informational and error messages are logged during the conversion.

Qualifiers

/FORMAT=*format-name*

Specifies the encoding format of the input or output file. The default input and output format is DDIF.

Input converters bundled with the CDA Base Services for DECwindows Motif for OpenVMS and the default file type for the file formats they support are as follows:

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

Output converters bundled with the CDA Base Services for DECwindows Motif for OpenVMS and the default file types for the file formats they support are as follows:

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

The CDA Converter Library is a layered product that offers several other document, graphics, image, and data table input and output converters. Independent software vendors also write CDA conforming applications and converters for the operating system. Contact your system manager for a complete list of converters available on your system.

Analysis Output Converter

The Analysis output converter produces an analysis of the intermediate representation of the input file. The analysis output file shows the named objects and values stored in the input file. Application programmers use an analysis output file for debugging purposes.

Application end users use an analysis output file to determine whether an input file contains references or links to multiple subfiles. Each subfile must be copied separately across a network because subfiles are not automatically included when an input file is transferred across the network.

You can search the analysis output file for all occurrences of the string "ERF_". The following example shows that the image file "griffin.img" is linked to the DDIF compound document that is the input file:

```
ERF_LABEL ISO LATIN1 "griffin.img" ! Char. string.
ERF_LABEL TYPE RMS_LABEL TYPE "$RMS:
ERF_CONTROL COPY_REFERENCE ! Integer = 1
```

Note that an analysis output file is intended as a programmer's tool. The coded information in the file is not intended for modification but rather to examine the content of a file. The previous example shows how you can search analysis output for references to linked files.

DDIF Input Converter

The DDIF input converter converts a DDIF input file to an intermediate representation that is subsequently converted to the specified output file format. The following list summarizes the data mapping, conversion restrictions, external file references, and document syntax errors relevant to the DDIF input converter:

Data mapping	The information in the DDIF input file maps directly to an intermediate representation.
Conversion restrictions	The DDIF input file does not lose any information when converted to the intermediate representation. However, if the DDIF input file is a newer version of the DDIF grammar than that understood by the DDIF input converter, data represented by the new grammar elements is lost.
External file references	Any external file references within the DDIF input file are converted to the intermediate representation. The DDIF input converter makes no attempt to resolve external references, although the converter kernel can if requested by the output converter.
Document syntax errors	A document syntax error in the DDIF input file causes a fatal input processing error. If the DDIF input converter encounters a document syntax error, the conversion stops and no further input processing occurs.

DDIF Output Converter

The DDIF output converter creates a DDIF output file from the intermediate representation of the input file. The following list summarizes the data mapping and conversion restrictions relevant to the DDIF output converter.

Data mapping	The information in the intermediate representation of the input file maps directly to the DDIF output file.
Conversion restrictions	The intermediate representation of the input file does not lose any information when converted to the DDIF output file.

DTIF Input Converter

The DTIF input converter converts a DTIF input file to an intermediate representation that is subsequently converted to the specified output file format. The following list summarizes the data mapping, conversion restrictions, external file references, and document syntax errors relevant to the DTIF input converter:

Data mapping	The information in the DTIF input file maps directly to an intermediate representation.
Conversion restrictions	The DTIF input file does not lose any information when converted to the intermediate representation. However, if the DTIF input file is a newer version of the DTIF grammar than that understood by the DTIF front end, data represented by the new grammar elements is lost.
External file references	Any external file references within the DTIF input file are converted to the intermediate representation. The DTIF input converter makes no attempt to resolve external references.
Document syntax errors	A document syntax error in the DTIF input file causes a fatal input processing error. If the DTIF input converter encounters a document syntax error, the conversion stops and no further input processing occurs.

DTIF Output Converter

The DTIF output converter converts the intermediate representation of the input file to a DTIF output file. The following list summarizes the data mapping, conversion restrictions, and external file references relevant to the DTIF output converter:

Data mapping	The information in the intermediate representation of the input file maps directly to the DTIF output file.
Conversion restrictions	The intermediate representation of the input file does not lose any information when converted to the DTIF output file.
External file references	The DTIF output converter converts external file references stored in the intermediate representation of the input file but makes no attempt to resolve external references.

Text Input Converter

The Text input converter converts a Text (ISO Latin1) input file to an intermediate representation that is subsequently converted to the specified output file format. The following list summarizes the data mapping, conversion restrictions, external file references, and document syntax errors relevant to the Text input converter:

Data mapping	The information in the text input file maps directly to an intermediate representation. Line breaks and form feeds are mapped to DDIF directives. One or more contiguous blank lines are interpreted as end-of-paragraph markers. If the text input file was entered as a DEC Multinational character set file on a character-cell terminal or terminal emulator, the following conversions occur:

	Original Character	**Converted Character**
	Concurrency sign	Diaeresis
	Capital OE ligature	Multiplication sign
	Capital Y with diaeresis	Capital Y with acute accent
	Small oe ligature	Division sign
	Small y with diaeresis	Y with acute accent
Conversion restrictions	The text input file does not lose any information when converted to the intermediate representation because no structure information is contained in a text file. All nonprinting characters are converted to space characters. For example, characters introducing ANSI escape characters are converted to space characters. There is no attempt to interpret ANSI escape sequences.	
External file references	Text files do not contain external file references.	
Document syntax errors	Text files do not contain syntax, so syntax errors are not reported by the Text input converter.	

Text Output Converter

The Text output converter converts the intermediate representation of the input file to a Text output file. The following list summarizes the data mapping and conversion restrictions relevant to the Text output converter:

Data mapping	All Latin1 text in the intermediate representation of the input file is converted to the text output file. When converting an input file to a text output file, you should be aware that text output files can contain only textual content and minimal formatting such as line feeds, page breaks, and tabs. The Text output converter preserves formatting information to the extent possible. Page coordinates convert to the nearest character cell (line, column) position.
Conversion restrictions	All graphics, images, and text attributes in the intermediate representation of the input file are lost when converted to the text output file. Because a monospace font is used, it is possible that some text may be lost due to overwriting to preserve the layout. It is also possible that lines can be truncated if the specified page width is smaller than the page width specified in the document's format information. Neither of these cases occur when you use the OVERRIDE_FORMAT processing option because, in that case, the document's format information is ignored.

PostScript Output Converter

The PostScript output converter converts the intermediate representation of the input file to a PostScript output file. The following list summarizes the data mapping and conversion restrictions relevant to the PostScript output converter.

Data mapping	The information in the intermediate representation of the input file maps directly to the PostScript output file.

Conversion restrictions	The intermediate representation of the input file does not lose any information when converted to the PostScript output file.

/MESSAGE_FILE=*filespec*
/NOMESSAGE_FILE (default)

Turns on message logging for document conversion. Messages output by the input and output converters are directed to the file specified with *filespec*. If filespec is not specified, messages are output to SYS$ERROR. The default is **/NOMESSAGE_FILE**.

/OPTIONS=*options-filename*

Specifies a text file that contains processing options applied to the input file and the output file during the conversion. The default file type for an options file is .CDA$OPTIONS.

Creating the Options File

You can create an options file prior to specifying the **CONVERT/DOCUMENT** command with the **/OPTIONS** qualifier. An options file is a text file with a default file type of .CDA$OPTIONS on the operating system.

The options file contains all the processing options for your input file format and your output file format. Processing options help ensure minimal changes when your input file is converted to a different output file format.

An options file is not required. Default processing options are applied automatically when you convert a file. However, you may require an options file if you need to use other than the default settings.

Use the following guidelines to create an options file:

- Begin each line of the options file with the keyword for the input or output format, followed by one or more spaces or tabs, or by a slash (/).

  For some file formats, such as DDIF and DTIF, there is an input converter and an output converter. You can restrict a processing option to only the input format or the output format by following the format keyword with _INPUT or _OUTPUT.

- Specify only one processing option on each line when there are several options for the same input or output format.

- Use uppercase and lowercase alphabetic characters, digits (0-9), dollar signs ($), and underscores (_) to specify the processing options.

- Use one or more spaces or tabs to precede values specified for a processing option.

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 because PostScript is available only as an output format. The value specified for `PAPER_HEIGHT` is in inches by default.

If the options file includes options that do not apply to the converters for a particular conversion, those options are ignored.

If you specify an invalid option for an input or output format or an invalid value for an option, you receive an error message. The processing options described in the following sections document any restrictions.

Processing Options for Analysis Output

The Analysis output converter supports the following options:

- *COMMENT DEFAULT_VALUES*

  Inserts a comment character (!) at the beginning of lines generated by default values. The comment prefix is also included on associated aggregate brackets and array parentheses where they may apply.

- *COMMENT INHERITED_VALUES*

  Inserts a comment character (!) at the beginning of lines generated by inherited values. The comment prefix is also included on associated aggregate brackets and array parentheses where they may apply.

- *TRANSLATE_BYTE_STRINGS*

  Overrides the default. For data of type BYTE STRING, the analysis output no longer displays the hexadecimal translation if all the characters in the byte string are printable characters (hex values 20 through 7E). This feature can be overridden by supplying the TRANSLATE_BYTE_STRINGS option.

- *IMAGE_DATA*

  Overrides the default. For the special case of byte string data for item DDIF $_IDU_PLANE_DATA (a bitmapped image), the analysis output previously included both a hexadecimal and an ASCII translation display, neither of which were of particular value to most users. With the new version, both displays will be replaced with the following comment:

  ```
  ! *** Bit-mapped data not displayed here ***
  ```

  To retain the hexadecimal display, supply the IMAGE_DATA option. Even with this option turned on, there will be no translation into ASCII.

- *INHERITANCE*

  Specifies that the analysis is shown with attribute inheritance enabled. Inherited attributes are marked as "[Inherited value.]" in the output. This option also causes external references to be imported into the main document.

Processing Options for Text Output

The Text output converter supports the following options:

- *ASCII_FALLBACK [ON,OFF]*

  Causes the Text output 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 output converter to put a message in the output file each time a nontext element is encountered in the intermediate representation of the input file. If this option is not specified, the default is OFF; if this option is specified without a value, the default is ON.

- *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 also 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 output 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 output 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 also 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 Output Converter

The PostScript output converter supports the following 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 (216 x 279 millimeters)
B	11 x 17 inches (279 x 432 millimeters)
C	17 x 22 inches (432 x 559 millimeters)

Keyword	Size
D	22 x 34 inches (559 x 864 millimeters)
E	34 x 44 inches (864 x 1118 millimeters)
LEDGER	11 x 17 inches (279 x 432 millimeters)
LEGAL	8.5 x 14 inches (216 x 356 millimeters)
LETTER	8.5 x 11 inches (216 x 279 millimeters)
LP	13.7 x 11 inches (348 x 279 millimeters)
VT	8 x 5 inches (203 x 127 millimeters)

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	Description
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 output converter should use 8-bit output. The default value is ON.

- *LAYOUT [ON,OFF]*

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

Domain Converter

You might create an options file containing processing options that apply to any CDA supported tabular file format for which there is an input converter. Data tables and spreadsheets are examples of tabular file formats.

To convert tabular input files to document output files, use the DTIF_TO_DDIF format name, followed by the processing options described in this section. Specify the DTIF_TO_DDIF processing options in addition to the processing options for a particular tabular input file format and a particular document output file format.

You might want to convert tabular input files to document output files so that you can include textual representations of tables in reports and other documents. You should be aware, however, that you lose cell borders, headers, grid lines, all formulas, and font types when converting a tabular input file to a document output file.

The domain converter supports the following options:

- *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. 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)
A5	148 x 210 millimeters (5.83 x 8.27 inches)
A	8.5 x 11 inches (216 x 279 millimeters)
B	11 x 17 inches (279 x 432 millimeters)
B4	250 x 353 millimeters (9.84 x 13.90 inches)
B5	176 x 250 millimeters (6.93 x 9.84 inches)
C	17 x 22 inches (432 x 559 millimeters)
C4	229 x 324 millimeters (9.01 x 12.76 inches)
C5	162 x 229 millimeters (6.38 x 9.02 inches)
D	22 x 34 inches (559 x 864 millimeters)
DL	110 x 220 millimeters (4.33 x 8.66 inches)
E	34 x 44 inches (864 x 1118 millimeters)
10x13_ENVELOPE	13 x 254 millimeters (15600 x 10 inches)
9x12_ENVELOPE	12 x 229 millimeters (14400 x 9 inches)
BUSINESS_ENVELOPE	9.5 x 105 millimeters (11400 x 4.13 inches)
EXECUTIVE	10 x 191 millimeters (12000 x 7.5 inches)
LEDGER	11 x 17 inches (279 x 432 millimeters)

Keyword	Size
LEGAL	8.5 x 14 inches (216 x 356 millimeters)
LETTER	8.5 x 11 inches (216 x 279 millimeters)
LP	13.7 x 11 inches (348 x 279 millimeters)
VT	8 x 5 inches (203 x 127 millimeters)

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 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 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 valid values for the *orientation* argument are as follows:

Keyword	Description
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.

Example

```
$ CONVERT/DOCUMENT/OPTIONS=MY_OPTIONS.CDA$OPTIONS -
```

```
_$ MY_INPUT.DTIF/FORMAT=DTIF MY_OUTPUT.DDIF/FORMAT=DDIF
```

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

CONVERT/RECLAIM

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 more information about the Convert/Reclaim utility, see the relevant section in the *VSI OpenVMS Record Management Utilities Reference Manual* [https://docs.vmssoftware.com/vsi-openvms-record-management-utilities-reference-manual/#CONV_RECLAIM_UTILITY] or online help.

Format

CONVERT/RECLAIM *filespec*

COPY

COPY — Creates a new file from one or more existing files.

Synopsis

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

Parameters

input-filespec[,...]

Specifies the name of an existing file to be copied. The asterisk (*) and the percent sign (%) wildcard characters 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 asterisk (*) 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 OpenVMS File Applications *[https://docs.vmssoftware.com/guide-to-openvms-file-applications/]* [https://docs.vmssoftware.com/guide-to-openvms-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 asterisk (*) and percent sign (%) 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* directory of the named directory. The **COPY** command does *not* copy any files from the named directory to the new directory. See the Examples section for examples of copying directory files.

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 absolute time, as a combination of absolute and delta times, or as one of the following keywords: BOOT, LOGIN, 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 relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT] or the online help topic Date.

/BLOCK_SIZE=*n*

Overrides the default block size (124) used by **COPY**. You can specify a value in the range of 1 through 2**31-1.

/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 relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#UIC_FORMAT_DIR].

/CONCATENATE (default)
/NOCONCATENATE

Creates one output file from multiple input files when you do not use the asterisk (*) or percent sign (%) 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 and 5 disks are concatenated in alphanumeric order. If you specify an asterisk (*) or percent sign (%) wildcard character 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 **Return**. 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. The asterisk (*) and the percent sign (%) wildcard characters 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; however, for RMS indexed and relative files, if the output file has fewer blocks allocated than the input file, the copy fails giving an RMS-E-EOF error.

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).
- Specify 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 relevant section in the *VSI OpenVMS Guide to System Security* [https://docs.vmssoftware.com/vsi-openvms-guide-to-system-security/#PROTECT_CODE_DETAILS].

/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 on or after the specified time. You can specify time as absolute time, as combination of absolute and delta times, or as one of the following keywords: BOOT, JOB_LOGIN, LOGIN, 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 about specifying time values, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT] or the online help topic Date.

/STYLE=*keyword*

Specifies the file name format for display purposes.

The valid keywords for this qualifier are CONDENSED and EXPANDED. Descriptions are as follows:

Keyword	Description
CONDENSED (default)	Displays the file name representation of what is generated to fit into a 255-length character string. This file name may contain a DID or FID abbreviation in the file specification.
EXPANDED	Displays the file name representation of what is stored on disk. This file name does not contain any DID or FID abbreviations.

The keywords CONDENSED and EXPANDED are mutually exclusive. This qualifier specifies which file name format is displayed in the output message, along with the confirmation if requested.

File errors are displayed with the CONDENSED file specification unless the EXPANDED keyword is specified.

See the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/] for more information.

/SYMLINK=*keyword*
/NOSYMLINK (default)

If the input file is a symbolic link, the file to which the symbolic link refers is the file that is copied.

The **/SYMLINK** qualifier indicates that any input symbolic link is copied.

If the file named in the command is a symlink, the command operates on the symlink target. The valid keywords for this qualifier are [NO]WILDCARD and [NO]ELLIPSIS. Descriptions are as follows:

Keyword	Description
NOWILDCARD	Indicates that symlinks are disabled during directory wildcard searches.
WILDCARD	Indicates that symlinks are enabled during wildcard searches.
NOELLIPSIS	Indicates that symlinks are matched for all wildcard fields except for ellipsis.
ELLIPSIS	Equivalent to WILDCARD (included for command symmetry).

/TRUNCATE (default)
/NOTRUNCATE

Controls whether the **COPY** command truncates an output file at the end-of-file (EOF) when copying it. This operation can only be used with sequential files.

By default, the actual size of the input file determines the size of the output file. If you select **/NOTRUNCATE**, 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 is written to verify that the record copied successfully and that the file can be read subsequently without error.

Note

Some hardware devices, such as TK50 tape drives, verify data integrity as part of their hardware function. For devices such as these, you do not need to use **/WRITE_CHECK**. For information about which devices provide automatic write checking, consult your hardware documentation.

Examples

1.
```
$ 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, DKA0:[MAL]NEW.DAT;1 being replaced
%COPY-S-COPIED, DKA0:[MAL]TEST.DAT;1 copied to DKA0:[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, DKA0:[MAL]A.TXT;2 copied to DKA0:[MAL]A.OLD;2 (1 block)
%COPY-S-COPIED, DKA0:[MAL]B.TXT;2 copied to DKA0:[MAL]B.OLD;2 (1 block)
%COPY-S-COPIED, DKA0:[MAL]G.TXT;2 copied to DKA0:[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 .TXT into new files. Each new file has the same file name as an existing file, but a file type .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, DKA0:[MAL]A.DAT;5 copied to DKA0:[MAL]C.DAT;11 (1 block)
%COPY-S-COPIED, DKA0:[MAL]B.MEM;2 copied to DKA0:[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, DKA0:[MAL]A.TXT;2 copied to DKA0:[MAL]TXT.SAV;1 (1 block)
%COPY-S-APPENDED, DKA0:[MAL]B.TXT;2 appended to DKA0:[MAL]TXT.SAV;1 (3
 records)
%COPY-S-APPENDED, DKA0:[MAL]G.TXT;2 appended to DKA0:[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 DKA1:[BACKUP]
```

   In this example, the **COPY** command copies the highest version of the file MASTER.DOC to the device DKA1. 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 DKA1 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.

14. ```
    $ COPY [SMITH]MONKEY.DIR [JONES]
    $ COPY [SMITH.MONKEY]*.* [JONES.MONKEY]*.*
    ```

    In this example, the **COPY** command creates the new empty directory [JONES.MONKEY] that is registered in the [JONES]MONKEY.DIR directory file. After the **COPY** command creates the new [JONES]MONKEY.DIR directory file, you can copy or create files in the [JONES.MONKEY] directory.

    The second **COPY** command in this example copies files from the [SMITH.MONKEY] directory to the [JONES.MONKEY] directory.

15. `$ COPY [SMITH]CATS.DIR [SMITH]DOGS.DIR`

    In this example, the **COPY** command creates the new empty directory file, called `[SMITH]DOGS.DIR`. Use this copy command to create a directory file that has the same attributes as the `[SMITH]CATS.DIR` file. This command example has the same effect as entering the command:

    ```
    $ CREATE/DIRECTORY [SMITH.DOGS]
    ```

16. ```
    $ COPY [SMITH]TIGER.DIR [SMITH.ANIMALS]
    $ COPY [SMITH.TIGER]*.* [SMITH.ANIMALS.TIGER]*.*
    $ DELETE [SMITH.TIGER]*.*;*
    $ SET SECURITY/PROTECTION=(WORLD:DELETE) TIGER.DIR
    $ DELETE TIGER.DIR;
    ```

    In this example, the **COPY** command creates the new empty directory file called `[SMITH.ANIMALS]TIGER.DIR`. The subsequent commands in this example then copy the files from the `[SMITH.TIGER]` directory to the `[SMITH.ANIMALS.TIGER]` directory, then delete the original `TIGER.DIR` directory file. Because `TIGER.DIR` is a directory file, you must specify a protection code of **DELETE** before you can delete the directory.

COPY/FTP

COPY/FTP — Transfers files between hosts with possibly dissimilar file systems over a TCP/IP connection by invoking the FTP utility.

Format

COPY/FTP ***input-filespec output-filespec***

Parameters

input-filespec

Specifies the name of an existing file (the source file) to be copied.

output-filespec

Specifies the name of the output file (the destination file) into which the input file is copied.

Description

The **COPY/FTP** command copies files to and from remote nodes using the File Transfer Protocol (FTP). The services provided by this command are a subset of the architectured features of FTP (see vendor documentation for usage of their supplied FTP program).

For OpenVMS to OpenVMS Transfers

If both machines support OpenVMS structured transfers, the **/BINARY**, **/ASCII**, and **/FDL** qualifiers will be ignored. The cooperating OpenVMS FTP client and server will automatically transfer the file with proper OpenVMS attributes.

COPY/FTP commonly supports the asterisk wildcard character (*) in remote file specifications.
    ```

Qualifiers

/ANONYMOUS

Causes an anonymous access to the remote node or nodes. **/ANONYMOUS** is the default remote access. The password passed to the remote node should be in the form of "user@fullyqualifiednodename".

/ASCII

Used to identify an ASCII file (text file). **/ASCII** is the default.

/BINARY

Required to identify binary files.

/FDL

This qualifier is optional. Causes interaction with an FDL (file definition language) file. If the file is being copied to the local OpenVMS system, are mote FDL file is sought and interpreted for the operation. If the file is being copied outside the local OpenVMS system, an FDL file is generated and copied in addition to the requested file. If the **/FDL** qualifier is specified and the vendor application does not support it, a warning message may be issued.

/LOG

Displays a message at SYS$OUTPUT when a file is transferred.

/NOSTRUVMS

Used to explicitly disable the negotiation of STRU OpenVMS transfers. Otherwise, some servers will immediately abort when negotiating the feature.

/PASSIVE=*option*

Controls whether the FTP client or server initiates the data connection. If you do not specify this qualifier, the Internet Protocol appropriate value is used. The values are: OFF for IPv4, ON for IPv6.

The following table describes the **/PASSIVE** options:

Option	Description
OFF	The FTP server initiates the data connection.
ON (default)	The FTP client initiates the data connection. This is often used where a firewall between the FTP client and server prevents the server from making an outbound connection. ON is the default value only if **/PASSIVE** is specified.

The underlying TCP/IP Networking product must recognize this qualifier and must support FTP passive in order for this qualifier to have an effect.

Note that the **/PASSIVE** qualifier is equivalent to the FTP PASV command.

/VERBOSE
/NOVERBOSE

Specifies whether all messages (including banner messages) are to be displayed on the terminal. By default, disables the display of the messages.

Examples

1. `$ COPY/FTP/FDL/ANON rms_indexed_file.idx -`
   `remotehst5::"/public/rms.idx.file"`

   This example transfers the OpenVMS RMS file `rms_indexed_file.idx` to the remote file `public/rms.idx.file` on `remotehst5` over a TCP/IP connection. Access to the remote host is anonymous and an FDL file is generated and copied along with `rms_indexed_file.idx`.

2. `$ COPY/FTP/VERBOSE sys$login:login.com -`
   `xdelta.zko.dec.com"username password"::sys$login:login.tmp`

   This example transfers the OpenVMS RMS file `sys$login:login.com` to the remote file `sys$login:login.tmp` over a TCP/IP connection while specifying the user name and password on the remote system.

3. `$ COPY/FTP/LOG RESULTS.LOG -`
   `_To: grad.uq.edu.au"JONES BYRONBAY"::DKA200$:[JONES.DATA]`

   In this example, the **COPY/FTP** command copies the file `RESULTS.LOG` to the file `DKA200$:[JONES.DATA]RESULTS.LOG` using the user account `JONES`, with password `BYRONBAY` on node `grad`, that is located in the `uq.edu.au` internet domain.

COPY/RCP

COPY/RCP — Copies files from host to host over a TCP/IP connection by invoking the RCP utility.

Format

COPY/RCP *input-filespec output-filespec*

Parameters

input-filespec

Specifies the name of an existing file (the source file) to be copied.

output-filespec

Specifies the name of the output file (the destination file) into which the input file is copied.

Description

The **COPY/RCP** command copies one or more files (or directory trees) to or from a remote host using the RCP utility.

The OpenVMS DCL commands for TCP/IP support the same remote file specification format as the DCL commands for DECnet network connections. Some implementations of the file transaction applications support file transfers in which both the source file and the destination file are remote file specifications.

The full format for a remote file specification is as follows:

```
node"username password account"::filename.ext
```

If a file resides on a system other than OpenVMS, enclose the name of the file in quotation marks. For example, to access a file named `/usr/users/user/Orders` on a Tru64 UNIX node named U32, you would use the following format for the file specification:

```
U32"user password"::"/usr/users/user/Orders"
```

Note that UNIX ® systems support case sensitive file specifications.

Qualifiers

/AUTHENTICATE

Specifies that Kerberos authentication should be used for acquiring access to the remote node.

/LOG

Displays a message in SYS$OUTPUT when a file is transferred.

/PRESERVE

Preserves the file protection codes.

/RECURSIVE

Requests a subdirectory copy operation.

/TRUNCATE= USERNAME

Truncates the user name to 8 characters.

/USERNAME=*username*

Optional qualifier that specifies the remote user name. The standard operation is to log in to a remote system using the same user name as at the local terminal. The command supports quoted parameters in the **`/USERNAME`** value.

Example

```
$ COPY/RCP local_file.c remotehst4"Smith smpw"::rem_file.c
```

This example copies `local_file.c` to `rem_file.c` on the remote host `remotehst4` over a TCP/IP connection.

COPY/RECORDABLE_MEDIA

COPY/RECORDABLE_MEDIA — The **`COPY/RECORDABLE_MEDIA`** (CDDVD) Utility allows users to create Compact Disk (CD) and Digital Versatile Disk (DVD) media directly on OpenVMS, using an optional optical disk recorder. CDDVD generates ISO/IEC 10149 Mode 1 (2048-byte blocks,

data) single-session optical media recordings. CDDVD supports the recording of various optical media formats, including CD Recordable (CD-R), CD Rewritable (CD-RW), DVD Recordable (DVD+R) and DVD Rewritable (DVD+RW) formats. For a successful recording operation, one or more of these formats must be available within the target optical disk recording device. Compatible recording media must also be loaded into the recording device. The **COPY/RECORDABLE_MEDIA** command opens the specified input disk image file or input master device and records the entire contents to the specified CD-R, CD-RW, DVD+R, and DVD+RW media formats.

Format

COPY/RECORDABLE_MEDIA*source-path-name target-path-name*

Parameters

input-filespec

Specifies the name of an existing file (the source file) to be copied.

source-path-name

This is the data source for the recording operation.

Specify the name of a disk file containing a disk image to be copied onto the target recording media, or the device name of the input device containing the disk volume master for the recording.

On OpenVMS systems, this is usually a Logical Disk (LD) Utility LDAu: device.

target-device-name

The device name of the target recordable media device.

This is usually the name of an ATAPI (DQcu:) SCSI (DKcu:), or USB (DNcu:): CD-R/RW or DVD+R/RW recording device, or both.

Description

The **COPY/RECORDABLE_MEDIA** command records the entire contents of the specified input disk image file or input device onto the media loaded into the specified output CD or DVD recording device.

The output media format is sensed automatically, and the utility automatically configures the recording appropriately for the particular target device and for the output media that are loaded.

You cannot record more than the capacity of the target media permits. Therefore, you need to select the size of the input disk image or the master appropriate for the capacity of the target media. The input data source must also be an even multiple of the sector size on optical media; the size of the input must be a multiple of four blocks.

The recording operation is independent of the input volume structure or input file data used for the master, and is based solely on the block-level contents of the specified input master.

Qualifiers

/BELL

Sounds an audible signal when the requested recording operation completes successfully.

/FORMAT[=*keyword*]
/NOFORMAT (default)

Requests that rewritable (RW) media be formatted or reformatted prior to use. This qualifier is required for writing to blank rewritable media or rewriting rewritable media.

If the target media cannot be formatted, this command qualifier is ignored.

If not specified, the appropriate keyword is automatically selected for the fastest formatting speed available for the target recording media.

Keywords for the **/FORMAT** Qualifier lists available keywords.

Keyword	Description
WAIT	Applies to DVD+RW. The default for the **/FORMAT** qualifier is not to wait for the formatting to complete because waiting is usually unnecessary and far slower. Selecting WAIT causes the entire format to run synchronously to completion before beginning the recording operation. The default is to: ● Operate asynchronously ● Perform background formatting ● Run both the media format operation and the recording operations concurrently
ERASE	Applies to CD-RW. The default for the **/FORMAT** qualifier is to perform a quick erasure because a full erasure is usually both unnecessary and far slower. Selecting ERASE causes the CD-RW rewritable disk to be entirely erased as part of the format operation. This erasure is performed and is completed before the recording operation begins. The default is to perform a quick erasure.

/LOG (default)
/NOLOG

Shows basic device information and the progress of the recording operation. Use **/NOLOG** to disable the normal output from the utility.

/SPEED

If you must use the lower-speed or poor-quality CD recording media, the **/SPEED** qualifier is often required for successful completion of the recording process. You might need to select a recording speed below the rated speed of the CD drive itself.

Specifically, you might need to select a recording speed that is compatible with both the CD drive and the CD recording media loaded in the drive.

The **/SPEED** qualifier accepts a single keyword for a requested device speed:

- 1X
- 2X
- 4X
- 8X
- 16X
- 32X
- MAXIMUM

The CDDVD utility attempts to match the requested speed to a speed that the device supports. Not all devices support all speeds, including the lowest speed, 1X, or the highest speed available. The default speed is the maximum speed that the target device supports. DVD+R/RW drives select the maximum recording speed based on information encoded on the media.

You need to specify this qualifier only under one of the following circumstances:

- When incompatibilities or recording errors are reported during a previous failed recording operation.
- If the CD media in use has a rated recording speed below the drive default recording settings.
- If CDDVD application, processor, or system I/O perform an constraints exist.

CD drives can select speeds faster than those supported by the particular media loaded in the drive. VSI recommends that you select only media that match the recording capabilities of the drive. In other words, do not attempt to exceed the recording speed limits of the particular CD media. Selecting faster media will not make a slow drive record any faster, and selecting faster speeds with slow media can trigger recording errors and corrupt media.

If the recording process fails during the recording operation, discard the write-once media and try a slower recording speed. Note that you can attempt to reformat and rerecord on rewritable media.

/VERIFY

Specifies that the contents of the output media be compared to the contents of the input source after the recording operation. Any data comparison errors detected are displayed.

/WRITE (default)
/NOWRITE

Allows you to test the system and device I/O throughput and the command syntax without recording on the target media.

If you specify **/NOWRITE** and if the target drive supports the underlying test-write hardware capability, all I/O operates as usual although **/NOWRITE** disables writing to the media.

/WRITE is the default, and causes the target optical media to be written.

Examples

1.
```
$ $ COPY/RECORDABLE_MEDIA -
$_ [/BELL] -
$_ [/DATA_CHECK=WRITE] -
```

```
$_ [/DIAGNOSTICS=(DETAILS,COMMANDS,ALL)] -
$_ [/EXTENSIONS[=(keywords)]] -
$_ [/[NO]LOG] -
$_ [/SPEED={1X|2X|4X|8X|16X|32X|MAXIMUM}] -
$_ source-path-name target-device-name
$
```

This example shows the generic format of the **COPY/RECORDABLE_MEDIA** command.

2.
```
$ COPY/RECORDABLE_MEDIA/FORMAT LDA1 DQA1
HP OpenVMS CD-R/RW and DVD+R/RW Utility V1.0-0
Copyright 1976, 2006 Hewlett-Packard Development Company, L.P.
Output device vendor: HP
Output device product name: DVD Writer 740b
Commencing media format operation
Formatting may require up to an hour
Output medium format: DVD+RW
Input data being read from: LDA1:
Input data size: 1200000 blocks
Starting operation at: 15:28:16
16 sectors written
30000 sectors written; estimated completion in 00:06:52; at 15:35:55
37000 sectors written; estimated completion in 00:06:54; at 15:36:07
46000 sectors written; estimated completion in 00:06:36; at 15:36:03
57000 sectors written; estimated completion in 00:06:08; at 15:35:51
71000 sectors written; estimated completion in 00:06:00; at 15:36:04
88000 sectors written; estimated completion in 00:05:26; at 15:35:56
110000 sectors written; estimated completion in 00:04:55; at 15:35:58
137000 sectors written; estimated completion in 00:04:12; at 15:35:56
171000 sectors written; estimated completion in 00:03:14; at 15:35:48
213000 sectors written; estimated completion in 00:02:10; at 15:35:48
266000 sectors written; estimated completion in 00:00:54; at 15:35:50
300000 sectors written; operation completed
Operation completed at: 15:35:47
Elapsed time for operation: 00:07:30
Synchronizing with output device cache
Processing completed
```

This example demonstrates recording the contents of `LDA1:` device onto the `DVD+RW` media loaded into device `DQA1:`.

CREATE

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 are cord 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 are cord. 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 codes (UICs)
- 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
/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 *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#UIC_FORMAT_DIR].

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

Specifies protection for the file.

- Specify the *ownership* parameter as system (S), owner (O), group (G), or world (W).
- Specify 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

Note

If you attempt to create a file with no access, the file is created with the system default RMS protection values. To create a file with no access, use the **SET SECURITY/PROTECTION** command.

For more information on specifying protection codes, see the relevant section in the *VSI OpenVMS Guide to System Security* [https://docs.vmssoftware.com/vsi-openvms-guide-to-system-security/#PROTECT_CODE_DETAILS].

/SYMLINK="*text*"

Creates a symbolic link containing the specified text without the enclosing quotation marks. If the created symbolic link is subsequently encountered during any file-name processing, the contents of the symbolic link are read and treated as a POSIX pathname specification. No previous version of the symbolic link can exist.

If the path is absolute (that is, it starts with a slash character), RMS attempts to translate its first field as a logical name.

/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**.

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.
```
$ SET DEFAULT DKA500:[TEST]
$ SET PROCESS /CASE=CASE_LOOKUP=SENSITIVE /PARSE_STYLE=EXTENDED
$ CREATE COEfile.txt
Ctrl/Z
$ CREATE COEFILE.TXT
Ctrl/Z
$ CREATE CoEfIlE.txt
Ctrl/Z
$ DIRECTORY

Directory DKA500:[TEST]
CoEfIlE.txt;
1COEFILE.TXT;
1COEfile.txt;1
```

In this example, DKA500 is an ODS-5 disk. If your process is set to CASE_LOOKUP=SENSITIVE and you create more than one file with the same name differing only in case, DCL treats subsequent files as new files and lists them as such.

CREATE/DIRECTORY

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

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.

Note

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. 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

/ALLOCATION=*n*

Specifies the initial number of blocks to be allocated to each of the specified directories. The default allocation is 1 block.

This qualifier is useful for creating large directories, for example MAIL.DIR;1. It can improve performance by avoiding the need for later dynamic expansion of the directory.

This qualifier does not apply to Files-11 ODS-1, ODS-3, or ODS-4 volumes.

/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 specify 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 relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#UIC_FORMAT_DIR].

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

Specifies protection for the directory.

- Specify the *ownership* parameter as system (S), owner (O), group (G), or world (W).
- Specify 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 codes, see the relevant section in the *VSI OpenVMS Guide to System Security* [https://docs.vmssoftware.com/vsi-openvms-guide-to-system-security/#PROTECT_CODE_DETAILS].

/VERSION_LIMIT[=*n*]

Sets the maximum number of versions that files in the directory can have. If you do not set a version limit, a value of 0 is used, indicating that the number of file versions is limited only to the Files-11 architectural limit of 32,767. If you change the version limit for a directory, the new version limit applies only to files created after the change has been made. When creating a file, if the total number of versions of that file name exceeds the specified version limit, then the file with the lowest version number is deleted from the directory without notification to the user.

The version limit set on a directory has no effect on the version limit set on a particular file in that directory. To set a version limit on a particular file in a directory, use the **SET FILE/VERSION_LIMIT[=*n*]** command.

To view the version limit on a directory, use the **DIRECTORY/FULL** command on a directory file and look at the File Attributes field of the output.

/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) -`
   `_$[KONSTANZ.SUB.HLP]`

   In this example, the **CREATE/DIRECTORY** command creates a subdirectory named `[KONSTANZ.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:[GOLDSTEIN]`

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

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

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

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

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

CREATE/FDL

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 more information about the Create /FDL utility, see the relevant section in the *VSI OpenVMS Record Management Utilities Reference Manual* [https://docs.vmssoftware.com/vsi-openvms-record-management-utilities-reference-manual/#CREATE_FDL_UTILITY] or online help.

Format

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

CREATE/MAILBOX (Alpha/Integrity servers Only)

CREATE/MAILBOX (Alpha/Integrity servers Only) — Creates a virtual mailbox named MBA*n* and assigns an I/O channel number to it. The **/MAILBOX** qualifier is required.

Format

CREATE/MAILBOX *logical-name*

Parameter

logical-name

Specifies a logical name for the new mailbox. The system creates the mailbox and sets the logical name to point to it.
   ```

Description

The **CREATE/MAILBOX** command creates a virtual mailbox.

Note

The following privileges are required:

- TMPMBX (temporary mailbox) to create a temporary mailbox (which is the default)
- CMEXEC (change mode to executive) to create a temporary mailbox (which is the default). Note that this requirement is temporary and will be removed in a future release.
- PRMMBX (permanent mailbox) to create a permanent mailbox (using the **/PERMANENT** qualifier)
- SYSNAM (system logical name) to place a logical name for a mailbox in the system logical name table
- GRPNAM (group logical name) to place a logical name for a mailbox in the group logical name table

Qualifiers

/BUFFER_SIZE=*n*

Specifies the number of bytes of system dynamic memory that can be used to buffer messages sent to the mailbox. If you do not specify **/BUFFER_SIZE** or specify it as 0, the operating system provides a default value from the DEFMBXBUFQUO system parameter.

/LOG
/NOLOG (default)

Displays the name of the new mailbox when it is created.

/MESSAGE_SIZE=*n*

Specifies the maximum size (in bytes) of a message that can be sent to the mailbox. The maximum value is 65535. If you do not specify **/MESSAGE_SIZE** or specify the value as 0, the operating system provides a default value from the DEFMBXMXMSG system parameter.

/PERMANENT

Specifies that the mailbox is to be permanent. By default, mailboxes are temporary.

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

Specifies protection for the mailbox.

- Specify the *ownership* parameter as system (S), owner (O), group (G), or world (W).
- Specify the *access* parameter as read (R), write (W), logical I/O (L), or physical I/O (P).

If no protection is specified, the mailbox template is used.

For more information about specifying protection codes, see the relevant section in the *VSI OpenVMS Guide to System Security* [https://docs.vmssoftware.com/vsi-openvms-guide-to-system-security/#PROTECT_CODE_DETAILS].

/TEMPORARY (default)

Specifies that the mailbox is temporary. By default, mailboxes are temporary unless you specify **/PERMANENT**.

Example

```
$ CREATE/MAILBOX/PERMANENT/MESSAGE_SIZE=512/LOG MY_MAILBOX
%CREATE-I-CREATED, MBA38: created
$ SHOW DEVICE MBA38/FULL
Device MBA38:, device type local memory mailbox, is online,
    record-oriented device, shareable, mailbox device.

 Error count              0  Operations completed               0
 Owner process           ""  Owner UIC                    [SYSTEM]
 Owner process ID  00000000  Dev Prot  S:RWPL,O:RWPL,G:RWPL,W:RWPL
 Reference count          0  Default buffer size              512
```

In this example, a permanent mailbox is created with the logical name MY_MAILBOX. The **SHOW DEVICE** command displays the full characteristics of the mailbox.

CREATE/NAME_TABLE

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

Synopsis

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 ($), and the underscore (_). Table names must be in uppercase letters; if you specify a name using lowercase letters, it will be converted to all uppercase. The table name is entered as a logical name in either the process directory logical name table (LNM$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.

For more information about logical name tables, see the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/] and *VSI OpenVMS System Manager's Manual, Volume 2: Tuning, Monitoring, and Complex Systems* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-2-tuning-monitoring-and-complex-systems/].

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:

Keyword	Description
CONFINE	Specifies that the table name and the logical names contained in the table are not copied into a spawned subprocess. This keyword can be 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	Specifies that no identical names (either logical names or names of logical name tables) can be created in an outer (less privileged) mode in the current directory. Unless you specify the NO_ALIAS attribute, the table can be "aliased" by an identical name created in an outer access mode. This attribute 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. This attribute applies to all types of logical name tables except clusterwide logical name tables. Whether or not you specify SUPERSEDE, the following conditions apply:

Keyword	Description
	• You cannot create a new clusterwide logical name table with the same name and access mode as an existing clusterwide logical name table until you delete the existing table. • If you specify a new clusterwide logical name table with the same name and access mode as an existing *local* logical name table, the new clusterwide logical name table is created, and the local table and its logical names are deleted. 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 either create (C) access to the parent table and write (W) access to the system directory or the SYSPRV privilege.

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).
- Specify the *access* parameter as read (R), write (W), create (C), or delete (D).

For more information on specifying protection codes, see the relevant section in the *VSI OpenVMS Guide to System Security* [https://docs.vmssoftware.com/vsi-openvms-guide-to-system-security/#PROTECT_CODE_DETAILS].

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 the table has no quota holder and 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.

Note

User-mode logical names are automatically deleted when invoking and exiting a command procedure.

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

CREATE/TERMINAL — Creates a window that emulates another terminal type.

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

Note

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

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 tot he 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 the **/NUMERIC_KEYPAD** qualifier.

/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-filespec*
/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 the **/OVERSTRIKE** qualifier.

/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 the **/APPLICATION_KEYPAD** qualifier.

/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 the **/INSERT** qualifier.

/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. You can log in from this window.

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.

Parameter	Description
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.

Examples

1. ```
   $ CREATE/TERMINAL=DECTERM/DETACH -
   _$ /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"`.

2. ```
   $ CREATE/TERMINAL=DECTERM -
   $_ /NOPROCESS -
   $_ /DEFINE_LOGICAL=(TABLE=LNM$GROUP,DBG$INPUT,DBG$OUTPUT)
   ```

   In this example, the command creates a DECterm with no associated process. The command defines `DBG$INPUT` and `DBG$OUTPUT` in the group table as the new terminal or the purposes of debugging a problem with a detached process that is subsequently created.

DEALLOCATE

DEALLOCATE — Makes an allocated device available to other processes (but does not deassign any logical name associated with the device). **DEALLOCATE** does not deallocate devices that are in use.

Synopsis

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 that is not in use. 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 that are not in use. 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

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 the logical name contains one or more trailing colons, you must append one additional colon to the **DEASSIGN** logical-name parameter (for example, type **DEASSIGN FILE::** to deassign the logical name 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**, **/JOB**, **/GROUP**, **/SYSTEM**, or **/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 delete a shareable logical name, you need write (W) access to the logical name table. To delete a shareable logical name table, you need write (W) access to the parent table and delete (D) access to the target logical name table.

To specify the access mode of the logical name you want to deassign, use the **/USER_MODE**, **/SUPERVISOR_MODE**, or **/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 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.

/CLUSTER_SYSTEM

You must be signed in to the SYSTEM account or have SYSNAM (system logical name) or SYSPRV (system) privilege to deassign a clusterwide logical name.

Deassigns a logical name from the LNM$SYSCLUSTER table.

/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 SYSNAM 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.

/LOG (default)
/NOLOG

/NOLOG overrides the default **/LOG** to suppress output of a fatal error that would be returned if the specified logical name were not found. When you specify **/NOLOG**, $STATUS is set to Success instead of to Fatal and no error message is output.

/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, 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

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*

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. Also, user-mode logical names are automatically deleted when invoking and exiting a command procedure.

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
$ SHOW LOGICAL COPY
   "COPY" = "USER_DISK:" (LNM$PROCESS_TABLE)
$ 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 DKA1: 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

DEASSIGN/ QUEUE — Deassigns a logical queue from a printer or terminal queue and stops the logical queue. The **DEASSIGN/QUEUE** command 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.

Note

Requires manage (M) access to the queue.

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

DEBUG — Invokes the OpenVMS Debugger. For a complete description of the OpenVMS Debugger, see the *VSI OpenVMS Debugger Manual* [https://docs.vmssoftware.com/vsi-openvms-debugger-manual/].

Format

DEBUG

Description

To get help on debugger commands from DCL level, type the following command:

```
$ HELP/LIBRARY=SYS$HELP:DBG$HELP DEBUG
```

(Heap Analyzer)

The Heap Analyzer provides a graphical representation of memory use in real time. This allows you to quickly identify inefficient memory usage in your application such as allocations that are made too often, memory blocks that are too large, fragmentation, or memory leaks.

For details on running the Heap Analyzer from within the debugger, see the relevant section in the *VSI OpenVMS Debugger Manual* [https://docs.vmssoftware.com/vsi-openvms-debugger-manual/#DBG_ANALYZER_CH].

On OpenVMS Integrity servers, the standalone Heap Analyzer is started within the kept debugger using the **START HEAP_ANALYZER** command.

On OpenVMS Alpha, the standalone Heap Analyzer is started within the kept debugger using the **RUN/HEAP** command.

Qualifiers

/CLIENT

Invokes the DEBUG client Motif interface. From the client, use the network binding string displayed by the server at startup to establish the connection. The first client to connect to the server is the primary client, and controls the number of secondary clients allowed to connect to the server.

/KEEP

Invokes the kept debugger. The kept debugger includes a Run/Rerun capability that allows you to debug an image multiple times or debug a series of distinct images without exiting the debugger.

Issuing the **DEBUG/KEEP** command is the only way to invoke the kept debugger.

/RESUME (default)

Reinvokes the non-kept debugger after a **Ctrl/Y** key sequence has interrupted the execution of a program you are debugging. The interrupted program must not have been linked with a **/NOTRACEBACK** qualifier on the **LINK** command.

If you issue the **DEBUG/RESUME** command without a previous **Ctrl/Y** key sequence, no action occurs.

/SERVER[= ([BINDING_INFO=*filespec*][,PROTOCOLS=(*protocol*[,...])])]

Invokes the DEBUG server. The DEBUG server allows up to 30 simultaneous connections from clients on the same or remote OpenVMS nodes, or from PC nodes running a supported Microsoft ®Windows ® platform.

If specified (optionally), the BINDING_INFO keyword specifies that the server binding identification strings are to be written to *filespec*. If not specified, no file is created.

(Optional) If specified, the PROTOCOLS keyword specifies which network protocols should be enabled for connection to the DEBUG server. Only the specified protocols are enabled. If not specified, all protocols are enabled. The *protocol* argument can be one or more of the following keywords:

ALL
[NO]DECNET
[NO]TCP_IP
[NO]UDP

The first client to connect to the server is the primary client. A client that connects to the server after the primary client establishes the connection is a secondary client. The primary client controls the number of secondary clients allowed to connect to the server.

The server displays a series of RPC binding strings that identify the port numbers through which the client can connect to the server. The port number appears in square brackets ([]) at the end of the identification strings.

When connecting from the client, the simplest port identification string consists of the node name of the server followed by the port number in square brackets. The following are all valid binding identification strings:

NODNAM[1234]
NCACN_IP_TCP:16.32.16.25[1112]

16.32.16.25[1112]
NCACN_DNET_NSP:63.1004[RPC20A020DD0001]

Note

You must hold the DBG$ENABLE_SERVER identifier in the rights database to be able to run the debug server. Exercise care when using the debug server. Once a debug server is running, anyone on the network has the ability to connect to the debug server.

Before granting the DBG$ENABLE_SERVER identifier, the system manager must create it by entering the command **DEBUG/SERVER** from an account with write access to the rights database. The system manager needs to do this only once. The system manager can then run the Authorize utility to grant the DBG$ENABLE_SERVER identifier to the user's account in the rights database.

Examples

1.
```
$ FORTRAN/DEBUG/NOOPTIMIZE WIDGET
$ LINK/DEBUG WIDGET
$ RUN WIDGET
                [ Debugger Banner and Version ]
%DEBUG-I-INITIAL, language is FORTRAN, module set to WIDGET
DBG>
```

   The **FORTRAN** and **LINK** commands both specify the **/DEBUG** qualifier to compile the program `WIDGET.FOR` with debugger symbol table information. Because the program has been compiled and linked with debug information, the debugger is automatically invoked by the image activator upon starting the program with the **RUN** command. No program code has yet been executed when the debugger is invoked.

2.
```
$ FORTRAN/DEBUG/NOOPTIMIZE WIDGET
$ LINK/DEBUG WIDGET
$ RUN/NODEBUG WIDGET
  NAME:  NAME:  NAME:
^Y
$ DEBUG/RESUME
             [ Debugger Banner and Version ]
  %DEBUG-I-INITIAL, language is FORTRAN, module set to WIDGET
  DBG>
```

   The **FORTRAN** and **LINK** commands both specify the **/DEBUG** qualifier to compile the program `WIDGET.FOR` with debugger symbol table information. The **RUN** command begins execution of the image `WIDGET.EXE`, which loops uncontrollably. **Ctrl/Y** interrupts the program, and the **DEBUG/RESUME** command gives control to the debugger.

3.
```
$ CC/DEBUG/NOOPTIMIZE ECHOARGS
$ LINK/DEBUG ECHOARGS
$ ECHO == "$ sys$disk:[]echoargs.exe"
$ DEBUG/KEEP
                    [ Debugger Banner and Version ]
DBG>
RUN/COMMAND="ECHO"/ARGUMENTS="fa sol la mi"
%DEBUG-I-INITIAL, language is C, module set to ECHOARGS
%DEBUG-I-NOTATMAIN, type GO to get to start of main program
DBG>
  .
```

```
   .
   .
DBG>
RERUN/ARGUMENTS="fee fii foo fum"
%DEBUG-I-INITIAL, language is C, module set to ECHOARGS
%DEBUG-I-NOTATMAIN, type GO to get to start of main program
DBG>
   .
   .
   .
DBG> RUN/ARGUMENTS="a b c" ECHOARGS
%DEBUG-I-INITIAL, language is C, module set to ECHOARGS
%DEBUG-I-NOTATMAIN, type GO to get to start of main program
DBG>
```

The **CC** and **LINK** commands both specify the **/DEBUG** qualifier to compile the program ECHOARGS.C with debugger symbol table information.

The symbol definition command defines a foreign command for use during the debugging session.

The **DEBUG/KEEP** command invokes the kept debugger.

The first **RUN** command uses the **/COMMAND** qualifier to specify a foreign command to invoke the image file and the **/ARGUMENTS** qualifier to specify a string of arguments.

The **RERUN** command reinvokes the same image file and uses the **/ARGUMENTS** qualifier to specify a new string of arguments.

The second **RUN** command specifies a new image file and a new string of arguments.

On Integrity server systems, start the Heap Analyzer within the kept debugger:

4.
```
$ debug/keep
DBG> run/heap 8queens
```

or, alternately:

5.
```
$ debug/keep
DBG> run 8queens
   .
   .
   .
DBG> deactivate break/all
DBG> deactivate watch/all
DBG> deactivate trace/all
DBG> start heap_analyzer
DBG> activate break/all
DBG> activate watch/all
DBG> activate trace/all
```

Using this method, you must first deactivate all watch points, breakpoints, and trace points before starting the heap analyzer with the **START HEAP_ANALYZER** command. This procedure prevents a potential race condition from occurring. After starting the heap analyzer, re-activate the breakpoints, watch points, and trace points.

On Alpha systems, start the Heap Analyzer within the kept debugger:

6.
```
$ debug/keep
```

```
DBG> run/heap 8queens
```

7.
```
$ DEBUG/SERVER=(PROTOCOLS=(TCP_IP,DECNET))
%DEBUG-I-SPEAK: TCP/IP: YES, DECnet: YES, UDP: NO
%DEBUG-I-WATCH: Network Binding: ncacn_ip_tcp:16.32.16.25[1112]
%DEBUG-I-WATCH: Network Binding: ncacn_dnet_nsp:63.1004[RPC20A020DD0001]
%DEBUG-I-AWAIT: Ready for client connection...
```

   The **DEBUG/SERVER** command establishes a connection to the debug server, requesting network protocols TCP/IP and DECnet. Note that the binding strings are saved in file TEMP.TMP. You can use the **TYPE** command to display the contents of TEMP.TMP.

DECK

DECK — Marks the beginning of an input stream for a command or program.

Format

DECK

Description

The **DECK** command marks the data that follows it as input for a command or program. The **DECK** command can be used only after a request to execute a command or program that requires input data.

In command procedures, this command is required when the first non-blank character in any data record in the input stream is a dollar sign. Also in command procedures, 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 indicator occurs for the current command level.

Qualifier

/DOLLARS[=*string*]

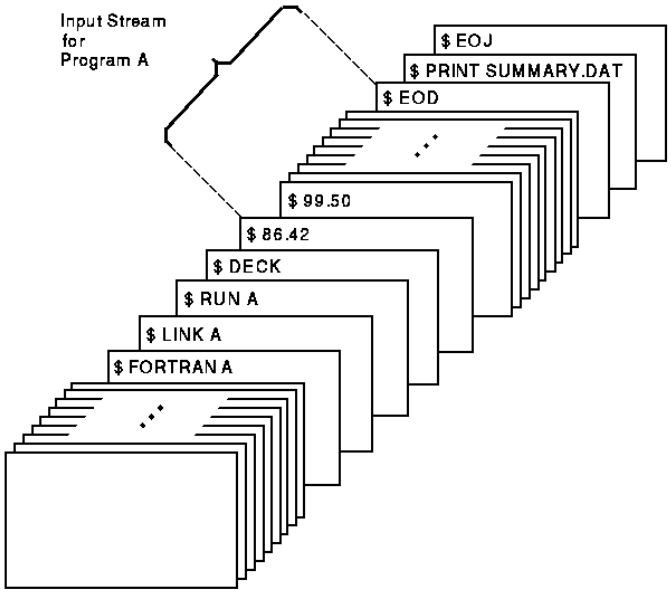
Sets the EOF indicator to the specified string of 1 to 15 characters. Specify a string if the input data contains one or more records beginning with the string $EOD. Enclose the string in quotation marks (" ") if it contains literal lowercase letters, multiple blanks, or tabs. If you do not specify **/DOLLARS** or if you specify **/DOLLARS** without specifying a string, you must use the **EOD** command to signal the end-of-file (EOF).

Note

A single dollar sign is not allowed as the end-of-deck or file indicator.

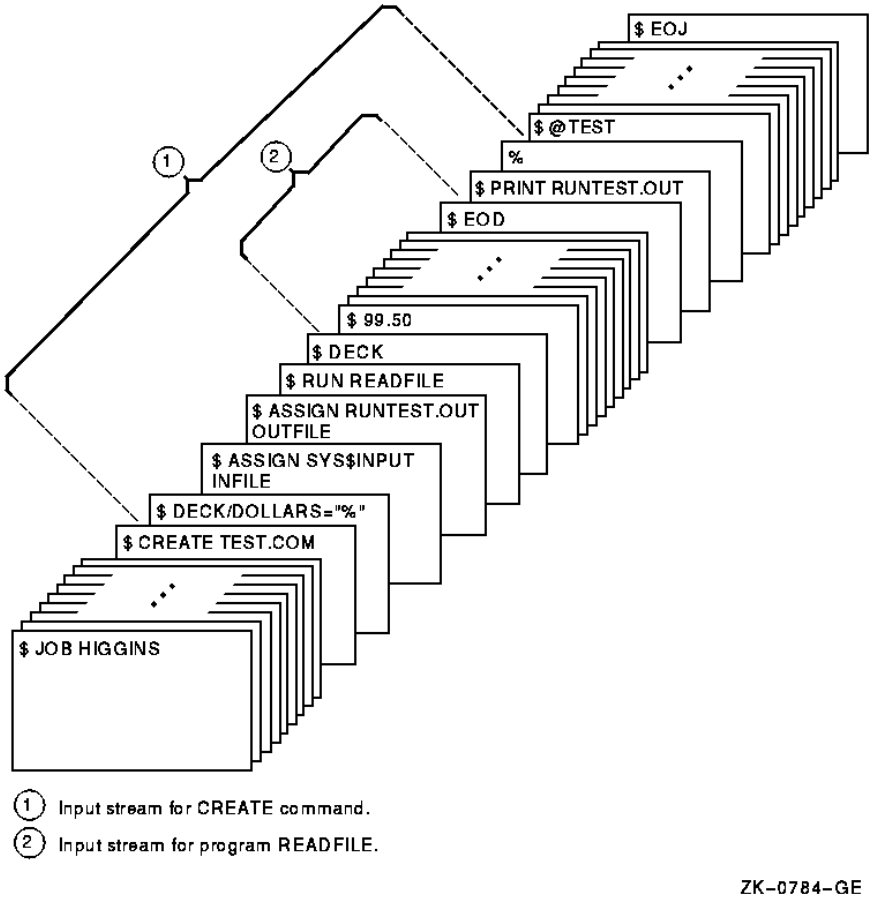
Examples

1.



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.



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.

DECRYPT

DECRYPT — Decrypts files previously encrypted with the **ENCRYPT** command. DES is the default algorithm unless otherwise specified with the **/KEY_ALGORITHM** qualifier. The key specified must match the algorithm (DES or AES), and the same key is used to decrypt as was used to encrypt; a symmetric key algorithm.

Format

DECRYPT*input-file key-name* [*qualifiers*]

Parameters

input-file

File names of the files to decrypt. If you use wildcard characters, do not include directory files or files with bad blocks.

key-name

Key name that was previously stored in the key storage table by the **ENCRYPT/CREATE_KEY** command.

Qualifiers

/BACKUP[=*time*]

Selects files according to the dates of their most recent backup.

This qualifier is relevant only when used with the **/BEFORE** or the **/SINCE** qualifier. In addition, do not use **/BACKUP** with **/EXPIRED** or **/MODIFIED**.

If you omit *time*, TODAY is used. For more information on time specifications, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT].

/BEFORE[=*time*]

Selects files that have a creation time before the time you specify.

If you omit *time*, TODAY is used. For more information on time specifications, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT].

/BY_OWNER[=*uic*]
/NOBY_OWNER

Selects files with the owner UIC you specify.

If you omit *uic*, the UIC of the current process is used. For more information on specifying UIC format, see the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#UIC_FORMAT_DIR].

/CONFIRM
/NOCONFIRM

Controls whether or not a confirmation request is displayed before each decryption, as follows:

Response	Meaning
YES	Decrypts the file
NO or **Return**	Does not decrypt the file (default)
QUIT or **Ctrl/Z**	Does not decrypt the file or any subsequent files
ALL	Decrypts the file plus all subsequent files

/DELETE
/NODELETE (default)

Controls whether or not the input files are deleted after the decryption operation is complete and the output file is written and closed.

/ERASE
/NOERASE

Controls whether or not the input files are erased with the data security pattern before being deleted. By default, the location in which the data was stored is not overwritten with the data security pattern. The **/ERASE** qualifier must be used with **/DELETE**.

/EXCLUDE=*file-spec*
/NOEXCLUDE

Excludes the specified files from the decryption operation. You can use wildcard characters. You do not need to enter an entire file specification. Any field that you omit defaults to the input file specification.

Because directory files are never encrypted, you need not specify them.

/EXPIRED[=*time*]

Selects files according to the dates on which they expire.

This qualifier is relevant only when used with the **/BEFORE** or the **/SINCE** qualifier. In addition, do not use **/EXPIRED** with **/BACKUP** or **/MODIFIED**.

If you omit *time*, TODAY is used. For more information on time specifications, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT].

/KEY_ALGORITHM=

1. DESCBC (default)
2. AES*mmmkkk*

Where *mmm* is the mode CBC, ECB, CFB, or OFB; and *kkk* is 128, 192, or 256 bits. Cipher Block Chaining (CBC) and Electronic Code Book (ECB) are 16-byte block modes, meaning blocks are padded to 16 bytes if necessary during encryption. The padding is removed during decryption. Cipher Feedback (CFB) and Output Feedback (OFB) are 8-bit character stream mode emulation, useful in data communications and where no padding is required. Note that **/KEY_ALGORITHM=AES** is a shortcut for specifying AESCBC128.

The algorithm by which the random key and the initialization vector are protected within the encrypted file. Specify the same algorithm (AES or DES) that you used to encrypt the file and create the key, if not, the default is DESCBC.

/MODIFIED[=*time*]

Selects files according to the dates on which they were last modified.

This qualifier is relevant only when used with the **/BEFORE** or the **/SINCE** qualifier. In addition, do not use **/MODIFIED** with **/BACKUP** or **/EXPIRED**.

If you omit *time*, TODAY is used. For more information on time specifications, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT].

/OUTPUT=*file-spec*

Alternate output file name for the decryption operation.

By default, each input file decrypted is written to a separate output file that is one version higher than that of the input file. When using the **/OUTPUT** qualifier, specify the parts of the file specification different from the defaults. You do not need to provide an entire file specification. Any field that you omit defaults to the input file specification.

/SHOW=(*keyword-list*)

Controls whether or not the following information about the decryption operation is displayed on SYS$COMMAND:

Keyword	Description
FILES	Displays input and output file names on SYS$COMMAND
STATISTICS	Displays the encryption stream statistics: • Bytes processed • Internal records processed • CPU time consumed within the encryption algorithm

/SINCE[=*time*]

Selects files that have a creation date before the time you specify.

If you omit *time*, TODAY is used. For more information on time specifications, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT].

/STATISTICS

Similar to **/SHOW**, except that **/STATISTICS** lists both files and statistics, whereas **/SHOW** can be customized to list only one or the other.

Examples

1. `$ DECRYPT BOSTON MYKEY`

   Decrypts the file name `BOSTON` using the DES key, `MYKEY`, and the DESCBC algorithm.

2. `$ DECRYPT CHIGAGO.ENC KEY2 /KEY=AESECB256 /OUT=CHICAGO.DEC`

   Decrypts the file named `CHICAGO.ENC` using the AES key, `KEY2`, and the `AESECB256` algorithm, renaming the decrypted output file to `CHICAGO.DEC`, the original plain text file.

DEFINE

DEFINE — Associates an equivalence name with a logical 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. The following rules apply:

- 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 the logical name translates to a logical name table name, any alphabetic characters in the name should all be uppercase.
- 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 sets of quotation marks ("" "") to denote actual quotation marks. When 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. The following rules apply:

- 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 sets of quotation marks to denote an actual quotation mark. Specifying more than one equivalence name for a logical name creates a search list. A logical name can have a maximum of 128 equivalence names.
- 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 multiple equivalence names to a single logical 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.

Description

The **DEFINE** command creates a logical name that represents one or more equivalence names. An equivalence name can be a device name, another logical name, a file specification, or any other string.

You can limit the use of a logical name to a process, a job, a group, an entire system, or an entire OpenVMS Cluster system. How you use a logical name depends on the table you created in it. You can specify a table with one of the following qualifiers: **/PROCESS**, **/JOB**, **/GROUP**, **/SYSTEM**, or **/TABLE**.

The first four qualifiers represent the process, job, group, or system logical name tables, respectively, whereas the **/TABLE** qualifier is used to specify any type of table. Furthermore, the **/TABLE** qualifier is the only one to use when specifying a clusterwide logical name table.

If you enter more than one of the qualifiers, only the last one entered is accepted. If you do not specify a table with one of the qualifiers, the logical name is added to your process logical name table.

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.

To create a logical name with no equivalence name (and therefore no indices), use the $CRELNM system service.

If you want to specify an ODS-5 file name as an equivalence name, see the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/].

For a complete description of logical names and logical name tables, except for their use in applications, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#LOGNAMES_CH]. For the use of logical names in applications, see the relevant section in the *VSI OpenVMS Programming Concepts Manual, Volume II* [https://docs.vmssoftware.com/vsi-openvms-programming-concepts-manual-volume-ii/#LOG_NAME]. For managing clusterwide logical names, see the relevant section in the *VSI OpenVMS Cluster Systems Manual* [https://docs.vmssoftware.com/vsi-openvms-cluster-systems/#CLUS_LOG_NAMES_H] manual. In this manual, see also the description of the lexical function F$TRNLNM, which is used to translate logical names.

Qualifiers

/CLUSTER_SYSTEM

You must be signed in to the SYSTEM account or have SYSNAM (system logical name) or SYSPRV (system) privilege to use this qualifier.

Defines a clusterwide logical name in the LNM$SYSCLUSTER table.

/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 SYSNAM (system logical name) 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:

Keyword	Description
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 write (W) access or SYSNAM (system logical name) 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, system, or clusterwide 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$PROCESS, LNM JOB, LNMGROUP, and LNM$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:

Keyword	Description
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). Also, user-mode logical names are automatically deleted when invoking and exiting a command procedure.

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

In this example, the **DEFINE** command associates the logical name CHARLIE with the directory name [CHARLES] on the disk XXX1. The **PRINT** command queues a copy of the file XXX1:[CHARLES]TEST.DAT to the system printer.

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""JAY_SABLE 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"JAY_SABLE 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.
```
$ DEFINE/TABLE=LNM$CLUSTER_TABLE FIRENZE FIRENZE::FIESOLE:[ETRUSCAN]
```

In this example, the **DEFINE** command equates FIRENZE to the directory specification FIRENZE::FIESOLE:[ETRUSCAN] and places both the new logical name (FIRENZE) and its

equivalence string (FIRENZE::FIESOLE:[ETRUSCAN]) in the default clusterwide table. The new logical name is automatically propagated to all nodes in the cluster.

9.
```
$  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 DKA1:
```

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 DKA1 and places the name in TABLE1. The logical name has the concealed attribute. Therefore, the logical name WORK_DISK is displayed in system messages.

10.
```
$ 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

DEFINE/CHARACTERISTIC — Assigns a numeric value to a queue characteristic. The **/CHARACTERISTIC** qualifier is required. If a value has been assigned to the characteristic, you must delete and redefine the characteristic to alter the assignment of the existing characteristic.

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. Only one characteristic name can be defined to each number.

characteristic-number

Assigns a number in the range 0 to 127 to the characteristic being defined.

Description

Note

Requires OPER (operator) privilege.

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.

Note

Prior to OpenVMS Version 6.0, the **DEFINE/CHARACTERISTIC** command allowed you to define more than one characteristic name to a number, although this capability was unsupported.

The **DEFINE/CHARACTERISTIC** command no longer allows you to define more than one characteristic name to a number; however, if your queue configuration requires you to have more than one characteristic name for a single number, you can define logical names to achieve the same result. For example, you might enter the following commands:

```
$ DEFINE/CHARACTERISTIC SECOND_FLOOR 2
$ DEFINE/SYSTEM/EXECUTIVE_MODE SALES_FLOOR SECOND_FLOOR
$ DEFINE/SYSTEM/EXECUTIVE_MODE SALES_DEPT SECOND_FLOOR
```

In this example, the characteristic name SECOND_FLOOR is assigned to the characteristic number 2. The logical names SALES_FLOOR and SALES_DEPT are then defined as equivalent to the characteristic name SECOND_FLOOR. As a result, the logical names SALES_FLOOR and SALES_DEPT are each equivalent to the characteristic name SECOND_FLOOR and the characteristic number 2. These logical names can be specified as the *characteristic-name* value for any **/CHARACTERISTIC=*characteristic-name*** qualifier.

In an OpenVMS Cluster environment, you must define the logical names on every node that requires them.

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** [https://docs.vmssoftware.com/vsi-openvms-dcl-dictionary-n-z/#BRASS_9000] and **SUBMIT** [https://docs.vmssoftware.com/vsi-openvms-dcl-dictionary-n-z/#BRASS_153] commands in the *VSI OpenVMS DCL Dictionary: N–Z* [https://docs.vmssoftware.com/vsi-openvms-dcl-dictionary-n-z/].

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 description of the **/CHARACTERISTICS** qualifier of the **INITIALIZE/QUEUE** command. See also the description of the **/CHARACTERISTICS** qualifier of the **SET QUEUE** [https://docs.vmssoftware.com/vsi-openvms-dcl-dictionary-n-z/#BRASS_83] and **START/QUEUE** [https://docs.vmssoftware.com/vsi-openvms-dcl-dictionary-n-z/#BRASS_137] commands in the *VSI OpenVMS DCL Dictionary: N–Z* [https://docs.vmssoftware.com/vsi-openvms-dcl-dictionary-n-z/].

The **DELETE/CHARACTERISTIC** command deletes a previously defined characteristic.

For more information on specifying queue characteristics, see the relevant section in the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/#BATCH_CHAR].

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

DEFINE/FORM — Assigns a numeric value and attributes to a print form name. The **/FORM** qualifier is required. To modify a form's name or number, you must delete and redefine the form. Values for any **DEFINE/FORM** qualifier can be modified by reentering the **DEFINE/FORM** command with different values, as long as the form name and number remain the same.

Format

DEFINE/FORMform-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 9999 to the form being defined. The DEFAULT form, which is defined automatically when the system is bootstrapped, is assigned number zero.

Description

Note

Requires OPER (operator) privilege.

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 in the *VSI OpenVMS DCL Dictionary: N–Z* [https://docs.vmssoftware.com/vsi-openvms-dcl-dictionary-n-z/#BRASS_9000].

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** and **/FORM_MOUNTED** qualifiers of the **INITIALIZE/QUEUE** command. See also the descriptions of the **/DEFAULT** and **/FORM_MOUNTED** qualifiers of the **SET QUEUE** [https://docs.vmssoftware.com/vsi-openvms-dcl-dictionary-n-z/#BRASS_83] and **START/QUEUE** [https://docs.vmssoftware.com/vsi-openvms-dcl-dictionary-n-z/#BRASS_137] commands in the *VSI OpenVMS DCL Dictionary: N–Z* [https://docs.vmssoftware.com/vsi-openvms-dcl-dictionary-n-z/].

For more information on how to use forms to control print jobs, see the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/].

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.

Option	Description
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.

For more information on device control modules, see the relevant section in the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/#DEVCTL].

/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. Jobs that request any of these forms will print on the same queue. If you want to modify the stock string associated with a form, you can do this only if the form is not referenced by any job or 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

DEFINE/KEY — Associates an equivalence string and a set of attributes with a key on the terminal keyboard.

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	---	---

Key Name	LK201	VT100-Series	VT52
Insert Here (**E2**)	Insert Here	---	---
Remove (**E3**)	Remove	---	---
Select (**E4**)	Select	---	---
Prev Screen (**E5**)	Prev Screen	---	---
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 **Return**). 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
  14-DEC-2001 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

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. The asterisk (*) and the percent sign (%) 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.

If an input-file specification parameter is a symbolic link, the symbolic link itself is deleted.

To delete more than one file, separate the file specifications with either commas (,) or plus signs (+).

Description

The **DELETE** command deletes one or more files from a mass storage disk volume. This command requires delete (D) access to the file and write (W) access to the parent directory.

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.

Note

Using this qualifier with the **DELETE/TREE** command results in an error.

/BEFORE[=*time*]

Selects only those files dated prior to the specified time. You can specify time as absolute time, as a combination of absolute and delta times, or as one of the following keywords: BOOT, LOGIN, 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 relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT] or the online help topic Date.

Note

Using this qualifier with the **DELETE/TREE** command results in an error.

/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 *VSI OpenVMS Guide to System Security* [https://docs.vmssoftware.com/vsi-openvms-guide-to-system-security/#PROTECT_CODE_DETAILS].

Note

Using this qualifier with the **DELETE/TREE** command results in an error.

/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 **Return**. 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.

Note

Using this qualifier with the **DELETE/TREE** command results in an error.

/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.

Note

Using this qualifier with the **DELETE/TREE** command results in an error.

/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. The asterisk (*) and the percent sign (%) wildcard characters 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.

Note

Using this qualifier with the **DELETE/TREE** command results in an error.

/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.

Note

Using this qualifier with the **DELETE/TREE** command results in an error.

/GRAND_TOTAL (Alpha/Integrity servers only)

Displays the total number of files and blocks or bytes deleted. The display is shown as blocks or bytes depending on the current default setting. You can use **SHOW PROCESS/UNITS** to display the current default. To change the default, execute the DCL command **SETPROCESS/UNITS=BYTES** or **SET PROCESS/UNITS=BLOCKS**.

/IGNORE=INTERLOCK (Alpha/Integrity servers only)

Allows you to mark a write-accessed file for deletion. This removes the file name entry, and the file is deleted when it is closed by the final user.

/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.

Note

Using this qualifier with the **DELETE/TREE** command results in an error.

/SINCE[=*time*]

Selects only those files dated on or after the specified time. You can specify time as absolute time, as a combination of absolute and delta times, or as one of the following keywords: BOOT, JOB_LOGIN, LOGIN, 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 about specifying time values, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT] or the online help topic Date.

Note

Using this qualifier with the **DELETE/TREE** command results in an error.

/STYLE=*keyword*

Specifies the file name format for display purposes while deleting files.

The valid keywords for this qualifier are CONDENSED and EXPANDED. Descriptions are as follows:

Keyword	Description
CONDENSED (default)	Displays the file name representation of what is generated to fit into a 255-length character string. This file name may contain a DID or a FID in the file specification.
EXPANDED	Displays the file name representation of what is stored on disk. This file name does not contain any DID or FID abbreviations.

The keywords CONDENSED and EXPANDED are mutually exclusive. This qualifier specifies which file name format is displayed in the output message, along with the confirmation if requested.

File errors are displayed with the CONDENSED file specification unless the EXPANDED keyword is specified.

See the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/] for more information.

/SYMLINK=*keyword*

The valid keywords for this qualifier are [NO]WILDCARD and [NO]ELLIPSIS. Descriptions are as follows:

Keyword	Description
WILDCARD	Indicates that symlinks are enabled during wildcard searches.
NOWILDCARD	Indicates that symlinks are disabled during directory wildcard searches.
ELLIPSIS	Equivalent to WILDCARD (included for command symmetry).
NOELLIPSIS	Indicates that symlinks are matched for all wildcard fields except for ellipsis.

If the file named in the **DELETE** command is a symlink, the command operates on the symlink itself.

Note

Using this qualifier with the **DELETE/TREE** command results in an error.

/TREE

Recursively deletes all files and sub directories excluding the parent directory.

You can specify only the following qualifiers with the **/TREE** qualifier:

/ERASE /GRAND_TOTAL /IGNORE /LOG
/STYLE

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 the asterisk (*) wildcards) must be included in each file specification.

4. ```
   $ DELETE /BEFORE=15-APR/LOG *.DAT;*
   %DELETE-I-FILDEL, DISK2:[MAIN]ASSIGN.DAT;1 deleted (5 block)
   %DELETE-I-FILDEL, DISK2:[MAIN]BATCHAVE.DAT;3 deleted (4 blocks)
   %DELETE-I-FILDEL, DISK2:[MAIN]BATCHAVE.DAT;2 deleted (4 blocks)
   %DELETE-I-FILDEL, DISK2:[MAIN]BATCHAVE.DAT;1 deleted (4 blocks)
   %DELETE-I-FILDEL, DISK2:[MAIN]CANCEL.DAT;1 deleted (2 blocks)
   %DELETE-I-FILDEL, DISK2:[MAIN]DEFINE.DAT;1 deleted (3 blocks)
   %DELETE-I-FILDEL, DISK2:[MAIN]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 [MEIER.TESTFILES]*.OBJ;*
DISK0:[MEIER.TESTFILES]AVERAG.OBJ;1, delete? [N]:Y
DISK0:[MEIER.TESTFILES]SCANLINE.OBJ;4, delete? [N]:N
DISK0:[MEIER.TESTFILES]SCANLINE.OBJ;3, delete? [N]:N
DISK0:[MEIER.TESTFILES]SCANLINE.OBJ;2, delete? [N]:N
DISK0:[MEIER.TESTFILES]WEATHER.OBJ;3, delete? [N]:Y
```

The **DELETE** command examines all versions of files with file type .OBJ in the subdirectory [MEIER.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 SECURITY/PROTECTION=(OWNER:DELETE) SUBTEST.DIR
$ 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 SECURITY/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 NODE12::"DISK1:DEAL.BIG"
$ DELETE NODE12::DISK1:DEAL.BIG;
```

Either of these **DELETE** commands can be used to delete the file DEAL.BIG on device ZZZ1 at remote node NODE12. 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. NODE12 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 [;]).

10.
```
$ DELETE/GRAND_TOTAL *.txt;*
%DELETE-I-TOTAL, 61 files deleted (274KB)
```

The output display in this example shows that 61 files were deleted for a total of 274KB. The process is currently set to display file sizes in bytes. To change future displays to show blocks, use the **SET PROCESS/UNITS=BLOCKS** command.

11.
```
$ DELETE/TREE $5$DKA100:[HOOPS...]*.*;*/LOG
%DELETE-I-FILDEL, $5$DKA100:[HOOPS]SMG_HP.EXE;2 deleted (32 blocks)
%DELETE-I-FILDEL, $5$DKA100:[HOOPS]TESTMSG.exe;4 deleted (32 blocks)
%DELETE-I-FILDEL, $5$DKA100:[HOOPS.DTM.EXAMPLES]TERMTABLE.TXT;1 -
```

```
 deleted - (16 blocks)
%DELETE-I-FILDEL, $5$DKA100:[HOOPS.DTM]EXAMPLES.DIR;1 deleted -
 (16 blocks)
%DELETE-I-FILDEL, $5$DKA100:[HOOPS]DTM.DIR;1 deleted (16 blocks)
%DELETE-I-TOTAL, 5 files deleted (112 blocks)
$
```

The **DELETE/TREE** command deletes all the files and sub directories recursively excluding the parent directory.

DELETE/BITMAP (Alpha/Integrity servers Only)

DELETE/BITMAP (Alpha/Integrity servers Only) — Enables the system manager to delete one or more active bitmaps to make memory resources available. If a minicopy bitmap is deleted, then former virtual unit members can be added only with a full copy operation. For more information about bitmaps, see the *VSI OpenVMS Volume Shadowing Guide* [https://docs.vmssoftware.com/vsi-openvms-volume-shadowing-guide/].

Format

DELETE/BITMAP ***n*[,*n*,...]**

Parameter

***n*[,*n*,...]**

Specifies the bitmap ID for one or more bitmaps to delete.

Qualifier

/LOG
/NOLOG (default)

Specifies whether to list each bitmap when it is deleted.

Description

Note

Requires ownership of the device or VOLPRO (volume protection) privilege.

Example

```
$ SHOW DEVICE /BITMAP DSA12

Device   BitMap    Size    Percent     Type of   Master  Active
 Name      ID     (Bytes)  Populated   Bitmap     Node
DSA12:  00020007     8364       0%    Minimerge  NODE1     Yes
        00040008     8364       0%    Minimerge  NODE2     Yes
```

```
$ DELETE/BITMAP 00020007
```

In this example, the **SHOW DEVICE** command output lists two bitmaps. The **DELETE** command deletes the bitmap with an ID of 00020007.

DELETE/CHARACTERISTIC

DELETE/CHARACTERISTIC — Deletes the definition of a queue characteristic previously established with the **DEFINE/CHARACTERISTIC** command. The **/CHARACTERISTIC** qualifier is required.

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.

Note

Requires OPER (operator) privilege.

To modify a characteristic's name or number, you must delete and redefine the characteristic.

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

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.

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 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.

Note

Requires manage (M) access to the queue, or delete (D) access to the specified jobs.

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 14-DEC-2001
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; therefore, 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

Entry  Jobname         Username             Status
-----  -------         --------             ------
   24  CHAPTER7        SMITH                Pending
   25  CHAPTER8        SMITH                Pending
$ DELETE/ENTRY=25 SYS$PRINT
```

   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

DELETE/FORM — Deletes a form (for printer or terminal queues) previously established with the **DEFINE/FORM** command. The **/FORM** qualifier is required.

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.

Note

Requires OPER (operator) privilege.

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 modify a form's name or number, you must delete and redefine the form. Values for any **DEFINE/FORM** qualifier can be modified by reentering the **DEFINE/FORM** command with different values, as long as the form name and number remain the same.

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

DELETE/INTRUSION_RECORD — Removes an entry from the break-in database.

Format

DELETE/INTRUSION_RECORD *source*

Parameter

source

Specifies the name of the device or the remote system where the user is attempting to log in. The source name can be presented in the syntax of another operating system domain, for example, one that is case sensitive or conflicts with DCL syntax rules. In such cases, you must enclose the source parameter in quotation marks.

Description

Use the **DELETE/INTRUSION_RECORD** command to remove an entry from the break-in database.

Note

Requires CMKRNL (change mode to kernel) and SECURITY privileges.

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.

Qualifiers

/NODE=(*node-name*[,...])

Deletes the node information relating to the specified nodes. If the specified nodes are the only nodes in the node information list, the intrusion record is also deleted.

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 user name is specified because none of the login failures occurred for valid users.

2. `$ DELETE/INTRUSION_RECORD "AV34C2/LC-2-10":FORGETFUL`

   In this example, the source of the break-in is a local terminal that is connected to a terminal server. To delete the record from the break-in database, you must enclose the terminal port name within quotation marks so that the operating system interprets the slash as a foreign character and not as a qualifier.

3. `$ DELETE/INTRUSION_RECORD NODE1::HAMMER`

   This command removes all intrusion entries generated from node NODE1 for user HAMMER.

4. ```
   $ DELETE/INTRUSION_RECORD/NODE=(CAPPY,INDI)
   $ SHOW INTRUSION
   NETWORK      SUSPECT       2  26-JUL-2001 08:51:25.66  BARNEY::HAMMER
       Node: TSAVO      Count:   2
   ```

   This command removes intrusion entries for the nodes CAPPY and INDI.

5. ```
   $ DELETE/INTRUSION_RECORD/NODE=FOOBAR
   $ SHOW INTRUSION
   NETWORK      SUSPECT       2  26-JUL-2001 08:51:25.66  BARNEY::HAMMER
       Node: TSAVO      Count:   2
   ```

   This command removes intrusion entries for the node FOOBAR.

6. ```
   $ DELETE/INTRUSION_RECORD/NODE=TSAVO
   $ SHOW INTRUSION
   %SHOW-F-NOINTRUDERS, no intrusion records match specification
   ```

   This command attempts to remove intrusion entries for node TSAVO, however there were no intrusion records for this node.

DELETE/KEY

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
 14-DEC-2001 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/MAILBOX (Alpha/Integrity servers Only)

DELETE/MAILBOX (Alpha/Integrity servers Only) — Deletes the specified mailbox.

Format

DELETE/MAILBOX *name*

Parameter

name

Specifies the name of the mailbox device (MBA *n*) or the logical name pointing to the mailbox to be deleted.

Description

Note

The command requires PRMMBX (permanent mailbox) privilege.

Qualifier

/LOG
/NOLOG (default)

Displays a notice when the mailbox is marked for deletion.

Example

```
$ SHOW LOGICAL MY_MBX
   "MY_MBX" = "MBA37:" (LNM$SYSTEM_TABLE)
$ SHOW DEVICE MBA37

Device                  Device           Error
 Name                   Status           Count
MBA37:                  Online               0
$ DELETE/MAILBOX/LOG MBA37
%DELETE-I-MBXDEL, Mailbox MBA37 has been marked for deletion
$ SHOW DEV MBA37
%SYSTEM-W-NOSUCHDEV, no such device available
```

This example shows the status of mailbox MBA37, which is pointed to by logical name MY_MBX, before and after it is deleted.

DELETE/QUEUE

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.

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:

Note

Requires manage (M) access to the queue.

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 still exist on any of its target queues count as references to the specified queue. Before you can delete the specified queue, 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/QUEUE/MANAGER

DELETE/QUEUE/MANAGER — Deletes a queue manager on a node or OpenVMS Cluster system. All queues and jobs managed by the specified queue manager are also deleted. You must first stop the queue manager. The **/NAME_OF_MANAGER** qualifier is required.

Format

DELETE/QUEUE/MANAGER/NAME_OF_MANAGER=*name*

Parameter

None.

Description

To delete a queue manager, use the following procedure:

Note

Requires OPER (operator) and SYSNAM (system logical name) privileges.

1. Stop the specified queue manager by using the **STOP/QUEUE/MANAGER/CLUSTER/NAME_OF_MANAGER=*name*** command.
2. Enter the **DELETE/QUEUE/MANAGER/NAME_OF_MANAGER** command, specifying the queue manager name.

Qualifier

/NAME_OF_MANAGER=*string*

Identifies the name of the queue manager to be deleted. The **/NAME_OF_MANAGER** qualifier is required. The required name value can be up to 31 characters long and can be a logical name.

Example

```
$ DELETE/QUEUE/MANAGER/NAME_OF_MANAGER=BATCH_MANAGER
```

The **DELETE/QUEUE/MANAGER/NAME_OF_MANAGER** command in this example deletes the queue manager named BATCH_MANAGER. The command removes all references to the specified queue manager from the shared master file of the queue database and deletes the queue and journal files associated with the BATCH_MANAGER's database.

DELETE/SYMBOL

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 KUDOS
%DCL-I-DELSYM, LOCAL symbol KUDOS has been deleted
```

   In this example, the **DELETE/SYMBOL** command deletes the symbol KUDOS 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

DEPOSIT — Replaces the contents of the specified locations in virtual memory and displays the new contents.

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.

Description

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 OpenVMS Debugger.

Note

Requires user-mode read (R) and write (W) access to the virtual memory location whose contents you wish to change.

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

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 %D2145876444
7FE779DC:  0000000000
```

```
$ DEPOSIT .=17
7FE779DC:  0000000017
$ CONTINUE
```

The **RUN** command executes the image MYPROG.EXE; subsequently, **Ctrl/Y** interrupts the program. Assuming that the initial defaults of the **/HEXADECIMAL** and **/LONGWORD** qualifiers are in effect, the **DEPOSIT** command places a longword value 17 (23 decimal) in virtual memory location 2145876444.

Because the **EXAMINE** command sets up a pointer to the current memory location, which in this case is virtual address 2145876444, 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

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. The asterisk (*) and the percent sign (%) 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. The asterisk (*) and the percent sign (%) 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
  /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 differences using the specified character. The **/CHANGE_BAR** qualifier displays output that depends on where the qualifier is placed. The following examples describe the result of **/CHANGE_BAR** qualifier placement.

The following placement displays the latest version of *input.file* with the pound sign (#) preceding any lines that differ from the preceding version of *input.file*:

```
$ DIFFERENCES input.file/CHANGE_BAR=#
```

The following placement displays *input.file;2* with the pound sign (#) preceding any lines that differ from *input.file;1*:

```
$ DIFFERENCES input.file;1  input.file;2 /CHANGE_BAR=#
```

The following placement displays *input.file;1* with the pound sign (#) preceding any lines that differ from *input.file;2*:

```
$ DIFFERENCES input.file;1/CHANGE_BAR=#  input.file;2
```

The following placement displays *input.file;1* with the percent sign (%) preceding any lines that differ from *input.file;2*, and also displays *input.file;2* with the pound sign (#) preceding any lines that differ from *input.file;1*:

```
$ DIFFERENCES input.file;1/CHANGE_BAR=% input.file;2/CHANGE_BAR=#
```

- 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 characters on a line to the right of (and including) a specified comment character.

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. Non-alphanumeric characters (such as `!` and `,`) must be enclosed in quotation marks. Multicharacter comment characters are not allowed. 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

If you specify the **/COMMENT_DELIMITER** qualifier, the **/IGNORE=COMMENTS** qualifier is implicitly also included.

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.

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 ;
.COB	* or / in the first column
.COM, .COR	!
.FOR	! anywhere and C, D, c, d in the first column
.HLP	!
.MAC, .MAR	;
.R32, .REQ	!

/EXACT

Use with the **/PAGE=SAVE** and **/SEARCH** qualifiers to specify a search string that must match the search string exactly and must be enclosed with quotation marks (" ").

If you specify the **/EXACT** qualifier without the **/SEARCH** qualifier, exact search mode is enabled when you set the search string with the Find (**E1**) key.

/HIGHLIGHT[=*keyword*]

Use with the **/PAGE=SAVE** and **/SEARCH** qualifiers to specify the type of highlighting you want when a search string is found. When a string is found, the entire line is highlighted. You can use the following keywords: BOLD, BLINK, REVERSE, and UNDERLINE. BOLD is the default highlighting.

/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.
CASE	Case of the text being compared.
COMMENTS	Data following a comment character. Use the **/COMMENT_DELIMITER** qualifier to designate one or more nondefault comment delimiters.
FORM_FEEDS	Form feed character.

Keyword	Item Ignored
HEADER[=*n*]	Defines *n* records of the file as header records, 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 header size and defaults to 2. A record containing only a single form feed is not counted in *n*.
SPACING	Extra spaces or tabs within data lines.
TRAILING_SPACES	Space and tab characters at the end of a data line.
WHITE_SPACE	All spaces and tab characters.
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. The following table shows the corresponding output for the various characters that are being translated:

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**, **/SEPARATED**, or **/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 .DIF. The asterisk (*) and the percent sign (%) wildcard characters are not allowed.

When you specify the **/OUTPUT** qualifier, you can control the defaults applied to the output file specification as described in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/]. The default output file type is .DIF.

/PAGE[=*keyword*]
/NOPAGE (default)

Controls the display of difference information on the screen.

You can use the following keywords with the **/PAGE** qualifier:

Keyword	Description
CLEAR_SCREEN	Clears the screen before each page is displayed.
SCROLL	Displays information one line at a time.
SAVE[=*n*]	Enables screen navigation of information, where *n* is the number of pages to store.

The **/PAGE=SAVE** qualifier allows you to navigate through screens of information. The **/PAGE=SAVE** qualifier stores up to 5 screens of up to 255 columns of information. When you use the **/PAGE=SAVE** qualifier, you can use the following keys to navigate through the information:

Key Sequence	Description
Up arrow key, **Ctrl/B**	Scroll up one line.
Down arrow key	Scroll down one line.
Left arrow key	Scroll left one column.
Right arrow key	Scroll right one column.
Find (**E1**)	Specify a string to find when the information is displayed.
Insert Here (**E2**)	Scroll right one half screen.
Remove (**E3**)	Scroll left one half screen.
Select (**E4**)	Toggle 80/132 column mode.
Prev Screen (**E5**)	Get the previous page of information.
Next Screen (**E6**), **Return**, **Enter**, **Space**	Get the next page of information.
F10, **Ctrl/Z**	Exit. Some utilities define these differently.
Help (**F15**)	Display utility help text.
Do (**F16**)	Toggle the display to oldest/newest page.
Ctrl/W	Refresh the display.

The **/PAGE** qualifier is not compatible with the **/OUTPUT** qualifier.

/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.

/SEARCH="*string*"

Use with the **/PAGE=SAVE** qualifier to specify a string that you want to find in the information being displayed. Quotation marks are required for the **/SEARCH** qualifier, if you include spaces in the text string.

You can also dynamically change the search string by pressing the Find key (**E1**) while the information is being displayed. Quotation marks are not required for a dynamic search.

/SEPARATED[=MASTER, REVISION]

Lists sequentially only the records from the specified file that contain differences. Use the MASTER keyword to list the differences in the first input file specified; use the REVISION keyword to list the differences in the second input file specified.

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*

Sets the length of the lines in the output display. If the display is to a terminal, the terminal line width is used as the default value. If the display is to a file, the default width is 132 characters.

/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.

/WRAP
/NOWRAP (default)

Use with the **/PAGE=SAVE** qualifier to limit the number of columns to the width of the screen and to wrap lines that extend beyond the width of the screen to the next line.

The **/NOWRAP** qualifier extends lines beyond the width of the screen and can be seen when you use the scrolling (left and right) features provided by the **/PAGE=SAVE** qualifier.

Examples

1.
```
$ DIFFERENCES EXAMPLE.TXT
************
File DISK1:[CHRIS.TEXT]EXAMPLE.TXT;2
     1   DEMONSTRATION
     2   OF V7.3 DIFFERENCES
     3   UTILITY
******
File DISK1:[CHRIS.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/ IGNORE=()/MERGED=1-
       DISK1:[CHRIS.TEXT]EXAMPLE.TXT;2-
       DISK1:[CHRIS.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.

2.
```
$ DIFFERENCES/PARALLEL/WIDTH=80/COMMENT_DELIMITER="V" EXAMPLE.TXT
--------------------------------------------------------------------------------
File DISK1:[CHRIS.TEXT]EXAMPLE.TXT;2  |  File DISK1:[CHRIS.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:[CHRIS.TEXT]EXAMPLE.TXT;2-
       DISK1:[CHRIS.TEXT]EXAMPLE.TXT;1
```

The **DIFFERENCES** command compares the same files as in Example 1, but ignores all characters following the first "V" on any line. The command also specifies that an 80-column parallel list of differences be displayed.

3.
```
$ DIFFERENCES/WIDTH=80/MODE=(HEX,ASCII) EXAMPLE.TXT/CHANGE_BAR
************
File DISK1:[CHRIS.TEXT]EXAMPLE.TXT;2
    1 ! DEMONSTRATION
    2 ! OF V7.3 DIFFERENCES
    3   UTILITY
************
************
File DISK1:[CHRIS.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 V7.3 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:[CHRIS.TEXT]EXAMPLE.TXT;2/CHANGE_BAR-
       DISK1:[CHRIS.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.

4.
```
$ 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

DIRECTORY — Provides a list of files or information about a file or group of files.

Format

DIRECTORY [*filespec*[,...]]

DIRECTORY/FTP *directory-spec*

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 name, 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.

The asterisk (*) and the percent sign (%) 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 (+).

directory-spec

Specifies the standard DECnet remote file specification. Use a quoted file string to preserve the case (for case sensitive systems such as UNIX) and to identify a foreign device/directory specification. See the **/FTP** qualifier for more information.

Description

The **DIRECTORY** command lists the files contained in a directory.

Note

Requires execute (E) access to look up files you know the names of, read (R) access to read or list a file or to use a file name with the asterisk (*) and the percent sign (%) wildcard characters to look up files.

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.

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.

You can also select other languages and formats that have been defined on your systems with international date and time formatting routines available in the run-time library. For further information, see the *VSI OpenVMS RTL Library (LIB$) Manual* [https://docs.vmssoftware.com/vsi-openvms-rtl-library-lib-manual/].

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. Access control entries (ACEs) that were created with the hidden option are displayed only if the SECURITY privilege is turned on. The **/ACL** qualifier overrides the **/COLUMNS** qualifier.

For further information, see the *VSI OpenVMS Guide to System Security* [https://docs.vmssoftware.com/vsi-openvms-guide-to-system-security/].

/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: BOOT, LOGIN, 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 relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT] or the online help topic Date.

/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 *VSI OpenVMS Guide to System Security* [https://docs.vmssoftware.com/vsi-openvms-guide-to-system-security/#PROTECT_CODE_DETAILS].

For further information, see the *VSI OpenVMS Guide to System Security* [https://docs.vmssoftware.com/vsi-openvms-guide-to-system-security/].

/CACHING_ATTRIBUTE

Displays the caching attributes of the selected files.

/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 specify more than one column and 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 creation, last modification, expiration, backup, effective, or recording 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:

Option	Description
ACCESSED	Specifies the last access date. See the Guide to OpenVMS File Applications *[https://docs.vmssoftware.com/guide-to-openvms-file-applications/]* [https://

Option	Description
	docs.vmssoftware.com/guide-to-openvms-file-applications/] for additional information.
ALL	Specifies all optional dates in the following order: creation, last modification, expiration, backup, effective, and recording.
ATTRIBUTES	Specifies the last attribute modification date. See the Guide to OpenVMS File Applications *[https://docs.vmssoftware.com/guide-to-openvms-file-applications/]* [https://docs.vmssoftware.com/guide-to-openvms-file-applications/] for additional information.
BACKUP	Specifies the last backup date.
CREATED	Specifies the creation date.
DATA_MODIFIED	Specifies the last data modification date. See the Guide to OpenVMS File Applications *[https://docs.vmssoftware.com/guide-to-openvms-file-applications/]* [https://docs.vmssoftware.com/guide-to-openvms-file-applications/] for additional information.
EFFECTIVE	Specifies the effective date the contents are valid (ISO9660).
EXPIRED	Specifies the expiration date.
MODIFIED	Specifies the last modification date.
RECORDING	Specifies the recording date on the media (ISO 9660).

/EXACT

Use with the **/PAGE=SAVE** and **/SEARCH** qualifiers to specify a search string that must match the search string exactly and must be enclosed with quotation marks (" ").

If you specify the **/EXACT** qualifier without the **/SEARCH** qualifier, exact search mode is enabled when you set the search string with the Find (**E1**) key.

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

Excludes the specified files from the **DIRECTORY** command. You can include a directory but not a device in the file specification.

The asterisk (*) and the percent sign (%) wildcard characters 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 (FID) number is displayed. By default, the FID is not displayed unless the **/FULL** qualifier is specified.

/FTP

Invokes the directory (*dir* or *ls*) operation of the FTP utility. The **DIRECTORY/FTP** command writes a listing of the contents of the specified remote directory to the local host over a TCP/IP connection by invoking the FTP utility.

The format is:

```
$ DIR/FTP nodename"username password"::directory_pathname
```

If the directory path name is omitted, the contents of the user's home directory are displayed. If only the node name is entered, the contents of the ANONYMOUS directory are displayed.

/FULL

Displays the following information for each file:

- File name
- File type
- Version number
- File identification number (FID)
- Number of blocks used
- Number of blocks allocated
- File owner's user identification code (UIC)
- Date of creation
- Date last modified and revision number
- Date of expiration
- Date of last backup
- Date of effective usage
- Date of recording on media
- File organization
- Shelved state
- Caching attribute
- File attributes

- Record format
- Record attributes
- RMS attributes
- Journaling information
- File protection
- Access control list (ACL)
- Client attribute
- 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.

/HIGHLIGHT[=*keyword*]

Use with the **/PAGE=SAVE** and **/SEARCH** qualifiers to specify the type of highlighting you want when a search string is found. When a string is found, the entire line is highlighted. You can use the following keywords: BOLD, BLINK, REVERSE, and UNDERLINE. BOLD is the default highlighting.

/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. The asterisk (*) and the percent sign (%) wildcard characters are not allowed.

If you enter the **/OUTPUT** qualifier with a partial file specification (for example, **/OUTPUT=[KIER]**), 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.

/PAGE[=*keyword*]
/NOPAGE (default)

Controls the display of directory information on the screen.

You can use the following keywords with the **/PAGE** qualifier:

Keyword	**Description**
CLEAR_SCREEN	Clears the screen before each page is displayed.
SCROLL	Displays information one line at a time.
SAVE[= *n*]	Enables screen navigation of information, where *n* is the number of pages to store.

The **/PAGE=SAVE** qualifier allows you to navigate through screens of information. The **/PAGE=SAVE** qualifier stores up to 5 screens of up to 255 columns of information. When you use the **/PAGE=SAVE** qualifier, you can use the following keys to navigate through the information:

Key Sequence	**Description**
Up arrow key, **Ctrl/B**	Scroll up one line.
Down arrow key	Scroll down one line.
Left arrow key	Scroll left one column.
Right arrow key	Scroll right one column.
Find (**E1**)	Specify a string to find when the information is displayed.
Insert Here (**E2**)	Scroll right one half screen.
Remove (**E3**)	Scroll left one half screen.
Select (**E4**)	Toggle 80/132 column mode.
Prev Screen (**E5**)	Get the previous page of information.
Next Screen (**E6**), **Return**, **Enter**, **Space**	Get the next page of information.
F10, **Ctrl/Z**	Exit. Some utilities define these differently.
Help (**F15**)	Display utility help text.

Key Sequence	Description
Do (**F16**)	Toggle the display to oldest/newest page.
Ctrl/W	Refresh the display.

The **/PAGE** qualifier is not compatible with the **/OUTPUT** 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.

/SEARCH="*string*"

Use with the **/PAGE=SAVE** qualifier to specify a string that you want to find in the information being displayed. Quotation marks are required for the **/SEARCH** qualifier, if you include spaces in the text string.

You can also dynamically change the search string by pressing the Find key (**E1**) while the information is being displayed. Quotation marks are not required for a dynamic search.

/SECURITY

Controls whether information about file security is displayed; using the **/SECURITY** qualifier is equivalent to using the **/ACL**, **/OWNER**, and **/PROTECTION** qualifiers together. ACEs that were created with the hidden option are displayed only if the SECURITY privilege is turned on.

For further information, see the *VSI OpenVMS Guide to System Security* [https://docs.vmssoftware.com/vsi-openvms-guide-to-system-security/].

/SELECT=(*keyword*[,...])

Allows you to select files for display. Choose one of the following keywords:

Keyword	Description
ACL NOACL	Displays files that have an associated ACL or files that do not (NOACL keyword).
CACHING_ATTRIBUTE =(*option*[,...])	Displays files that have the specified caching attribute. Possible options are: NO_CACHING WRITETHROUGH
FILE=(*option*[,...])	Displays portions of the file specification. The **/SELECT=FILE** qualifier is used to turn off specific portions by explicit or implicit specification of the options. Possible options are: [NO]NODE

<table>
<tr><th>Keyword</th><th colspan="2">Description</th></tr>
<tr><td></td><td colspan="2">[NO]DEVICE
[NO]DIRECTORY
[NO]NAME
[NO]TYPE
[NO]VERSION

/SELECT=FILE qualifier cannot be used with the /FULL qualifier.</td></tr>
<tr><td>ONLINE
NOONLINE</td><td colspan="2">Displays files that are online or shelved.</td></tr>
<tr><td>PRESHELVED
NOPRESHELVED</td><td colspan="2">Displays files that are preshelved or not preshelved.</td></tr>
<tr><td>SHELVABLE
NOSHELVABLE</td><td colspan="2">Displays files that are shelvable or not shelvable.</td></tr>
<tr><td rowspan="5">SIZE=(option[,...])</td><td colspan="2">Displays files according to their size. Possible options are:</td></tr>
<tr><td>Option</td><td>Description</td></tr>
<tr><td>MAXIMUM= n</td><td>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.</td></tr>
<tr><td>MINIMUM= n</td><td>Displays files that have blocks equal to or greater than the value of n. Use with MAXIMUM=n to specify a size range for files to be displayed.</td></tr>
<tr><td>(MINIMUM= n, MAXIMUM= n)</td><td>Displays files whose block size falls within the specified MINIMUM and MAXIMUM range.</td></tr>
<tr><td></td><td>UNUSED[=n]</td><td>Displays a file only if the difference between the used portion of a file and the allocated size of a file exceeds the disk's cluster size. If a value is specified, any file with unused space exceeding that value is displayed.</td></tr>
<tr><td>VERSION=(option[,option])
(Alpha/Integrity servers Only)</td><td colspan="2">Displays all files with version numbers that fall within the range specified by one or both of the following options:

MINIMUM=number
MAXIMUM=number</td></tr>
</table>

/SHELVED_STATE

Displays whether the file is shelved, preshelved, or online.

/SINCE[=*time*]

Selects only those files dated on or 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: BOOT, JOB_LOGIN, LOGIN, 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 relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT] or the online help topic Date.

/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:

Option	Description
ALL	Lists the file size both in blocks allocated and blocks used.
ALLOCATION	Lists the file size in blocks allocated.
UNITS[=*option*]	Allows you to override the current default specified by **SET PROCESS/UNITS** so that you can display file size in your choice of blocks or bytes. The following keywords are valid options with the UNITS keyword: BLOCKS, BYTES. If you specify UNITS with no option, the default value is not changed.
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.

/STYLE=*keyword*[,*keyword*]

Specifies the file name format for display purposes while displaying directory contents.

The valid keywords for this qualifier are CONDENSED and EXPANDED. Descriptions are as follows:

Keyword	Description
CONDENSED (default)	Displays the file name representation of what is generated to fit into a 255-length character string. This file name may contain a DID or FID abbreviation in the file specification.
EXPANDED	Displays the file name representation of what is stored on disk. This file name does not contain any DID or FID abbreviations.

If both CONDENSED and EXPANDED keywords are specified, then the file specifications are displayed in two columns. The column size is dependent on the display width, and the file names wrap within their respective columns.

File errors are displayed with the CONDENSED file specification unless the EXPANDED keyword is specified.

See the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/] for more information.

/SYMLINK=*keyword* (default)
/NOSYMLINK

If an input-file specification parameter is a symbolic link, the displayed file attributes are those of the symbolic link itself. If any file attribute is requested, then the contents of the symbolic link are also displayed, with an arrow appearing between the file name and the contents (for example, LINK.TXT -> FILE.TXT).

The **/NOSYMLINK** qualifier indicates that if an input file specification is a symbolic link, then the file attributes of the file to which the symbolic link refers are displayed; the displayed name is still the name of the symbolic link itself.

The valid keywords for this qualifier are [NO]WILDCARD, [NO]ELLIPSIS, and [NO]TARGET. Descriptions are as follows:

Keyword	Description
WILDCARD	Indicates that symlinks are enabled during wildcard searches.
NOWILDCARD	Indicates that symlinks are disabled during directory wildcard searches.
ELLIPSIS	Equivalent to WILDCARD (included for command symmetry).
NOELLIPSIS	Indicates that symlinks are matched for all wildcard fields except for ellipsis.
TARGET	Indicates that if the named file is a symlink, then the symlink is followed to operate on the symlink target.
NOTARGET	Indicates that the command operates on the named file whether it is an ordinary file or a symlink.

If the file named in the **DIRECTORY** command is a symlink, the command by default operates on the symlink itself.

/TIME[=*option*]
/NOTIME (default)

The same as the **/DATE** qualifier: includes the backup, creation, expiration, or modification time for each specified file; the default is the **/NOTIME** qualifier. If you use the **/TIME** qualifier without an option, the creation time is provided. Possible options are as follows:

Option	Description
ALL	Specifies creation, expiration, backup, and last modification times.
BACKUP	Specifies the last backup time.
CREATED	Specifies the creation time.
EFFECTIVE	Specifies the effective time the contents are valid.
EXPIRED	Specifies the expiration time.
MODIFIED	Specifies the last modification time.
RECORDING	Specifies the recording time on the media.

/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:

Keyword	Description
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=*n*** 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 on systems prior to OpenVMS Version 6.0; the default is 7 characters on OpenVMS

Keyword	Description
	Version 6.0 systems or higher. If the file size exceeds the length of the size field, the field is filled with asterisks.

/WRAP
/NOWRAP (default)

Use with the **/PAGE=SAVE** qualifier to limit the number of columns to the width of the screen and to wrap lines that extend beyond the width of the screen to the next line.

The **/NOWRAP** qualifier extends lines beyond the width of the screen and can be seen when you use the scrolling (left and right) features provided by the **/PAGE=SAVE** qualifier.

Examples

1.
```
$ DIRECTORY AVERAGE.*

Directory DISK$DOCUMENT:[SOUDER]

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:[SLOUGH]

AVERAGE.EXE;6       6        19-DEC-2001 15:43:02.10 (RE,RE,RWED,RE)
AVERAGE.FOR;6       2        19-DEC-2001 10:29:53.37 (RE,RE,RWED,RE)
AVERAGE.LIS;4       5        19-DEC-2001 16:27:27.19 (RE,RE,RWED,RE)
AVERAGE.OBJ;6       2        19-DEC-2001 16:27:44.23 (RE,RE,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 DISK$GRIPS_2:[VMS.TV]DEMO.EXE

Directory DISK$GRIPS_2:[VMS.TV]

DEMO.EXE;1                      File ID:  (36,11,0)
Size:           390/390         Owner:    [0,0]
Created:  12-NOV-2001 11:45:19.00
Revised:  14-DEC-2001 15:45:19.00 (34)
Expires:
<None specified>
Backup:   28-NOV-2001 04:00:12.22
Effective:
<None specified>
Recording:
<None specified>
```

```
File organization:  Sequential
Shelved state:      Online
Caching attribute:  Writethrough
File attributes:    Allocation: 390, Extend: 0, Global buffer count: 0,
                    Version limit: 0, Backups disabled, Not shelvable
Record format:      Fixed length 512 byte records
Record attributes:  None
RMS attributes:     None
Journaling enabled: None
File protection:    System:RE, Owner:RE, Group:RE, World:RE
Access Cntrl List:  None
Client attributes:  None

Total of 1 file, 390/390 blocks.
```

The example illustrates the **DIRECTORY/FULL** command.

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:[TAMBA]*.COM`

   The **DIRECTORY** command in this example lists all versions of all files with the file type COM in the directory TAMBA 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=14-DEC-2001:01:30/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 December 14, 2001, 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.

9. `$ DIRECTORY/SHELVED_STATE`

   ```
   Directory MYDISK:[THOMPSON]

   MYFILE.TXT;2            Online
   NOT_SHELVED.TXT;1       Online
   ```
   ```

```
SHELVED.TXT             Shelved

Total of 3 files.
```

The **DIRECTORY** command in this example lists all the files in a directory and shows whether a file is shelved, preshelved, online, or remote.

10.
```
$ DIRECTORY *.PS

Directory MYDISK:[TEST]

REPORT.PS;1            1197

Total of 1 file, 1197 blocks.

$ DIRECTORY/SIZE=UNITS=BYTES *.PS

Directory $1$DKC600:[TEST]

REPORT.PS;1           598KB

Total of 1 file, 598KB
```

By default, the first **DIRECTORY** command displays the file size in blocks. The second **DIRECTORY** command specifies that the file size be displayed in bytes.

DISABLE AUTOSTART

DISABLE AUTOSTART — Disables the autostart feature on a node for all autostart queues managed by the specific queue manager. By default, this command uses the **/QUEUES** qualifier. For more information on autostart queues, see the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/#_6017QUEUES].

Format

DISABLE AUTOSTART[/QUEUES]

Description

The **DISABLE AUTOSTART/QUEUES** command notifies the queue manager to perform the following tasks on the affected node:

- Mark all of the queue manager's autostart queues as "stop pending" in preparation for a planned shutdown.
- Prevent any of the queue manager's autostart queues from failing over to the node.
- Upon completion of any jobs currently executing on any of that queue manager's autostart queues, force the queue to fail over to the next available node in the queue's failover list (if any) on which autostart is enabled. For information on failover lists for autostart queues, see the **/AUTOSTART_ON** qualifier for the **INITIALIZE/QUEUE** command.

Autostart queues on the node that do not have a failover list, or for which no failover node is enabled for autostart, are stopped upon completion of any current jobs. These stopped queues remain

activated for autostart. The queue manager will restart these stopped autostart queues when the **ENABLE AUTOSTART** command is entered for the affected node or a node to which the queue can fail over.

By default the command affects the node on which it is entered. Specify the **/ON_NODE** qualifier to disable autostart on a different node.

The **DISABLE AUTOSTART/QUEUES** command is included in the node shutdown command procedure SHUTDOWN.COM. If you shutdown a node without using SHUTDOWN.COM, and the node is running autostart queues, you might want to enter the **DISABLE AUTOSTART/QUEUES** command first.

The **DISABLE AUTOSTART/QUEUES** command only affects autostart queues.

Qualifiers

/NAME_OF_MANAGER=*name*

Specifies the name of the queue manager controlling the autostart queues you want to disable. The qualifier allows the autostart feature to be used differently for different sets of queues.

If the **/NAME_OF_MANAGER** qualifier is omitted, the default queue manager name SYS$QUEUE_MANAGER is used. For more information on multiple queue managers, see the relevant section in the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/#CREATE_ADD_QM].

/ON_NODE=*nodename*

Specifies a node in an OpenVMS Cluster system. Use this qualifier to disable autostart on a node other than the one from which you enter the command.

/QUEUES

Specifies that autostart is to be disabled for queues. This qualifier is used by default.

Examples

1.
```
$ INITIALIZE/QUEUE/BATCH/START/AUTOSTART_ON=SATURN:: BATCH_1
$ ENABLE AUTOSTART/QUEUES
   .
   .
   .
$ DISABLE AUTOSTART/QUEUES
```

   In this example, the **INITIALIZE/QUEUE** command creates an autostart queue BATCH_1, capable of executing on node SATURN. The **/START** qualifier activates the queue for autostart. The **ENABLE AUTOSTART/QUEUES** command (executed on node SATURN) enables autostart on the node, causing the queue (and any other active autostart queues on the node) to begin executing jobs.

   The **DISABLE AUTOSTART** command (executed on node SATURN) stops autostart queues on the node, and prevents any queues from failing over to the node.

   This command only affects queues managed by the default queue manager SYS$QUEUE_MANAGER because the **/NAME_OF_MANAGER** qualifier is not specified.

Because BATCH_1 is set up to run only on one node, the queue cannot fail over to another node and therefore is stopped; however, the queue remains active for autostart and will be started when the **ENABLE AUTOSTART** command is entered for node SATURN. No **START/QUEUE** command is needed to restart BATCH_1 unless autostart of the queue is deactivated with the **STOP/QUEUE/NEXT** or **STOP/QUEUE/RESET** command.

2. `$ DISABLE AUTOSTART/QUEUES/ON_NODE=JADE`

   The **DISABLE AUTOSTART/QUEUES** command in this example disables autostart on the OpenVMS Cluster node JADE. This command can be entered from any node in the cluster.

DISCONNECT

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.

Format

DISCONNECT

Description

Use the **DISCONNECT** command to disconnect a physical terminal from a virtual terminal and its associated process.

Note

Requires that your physical terminal is connected to a virtual terminal.

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

DISMOUNT — Closes a mounted disk or tape volume for further processing and deletes the logical name associated with the device.

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 command requires the GRPNAM (group logical name) and SYSNAM (system logical name) privileges to dismount group and system volumes.

DUMP

DUMP — Displays the contents of a file, a directory, a disk volume, a magnetic tape volume, or a CD-ROM volume in decimal, hexadecimal, octal format, ASCII, or formatted data structures. This command can be used to generate process dumps.

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 the asterisk (*) and the percent sign (%) wildcard characters.

Files-11 C/D format standards have been implemented on mounted and foreign mounted volumes.

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. The asterisk (*) and the percent sign (%) wildcard character 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.

Dumping Processes

If you use the **/PROCESS** qualifier, the **DUMP** command attempts to generate a process dump file.

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

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:

Option	Description
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 blocks 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.

/DESCRIPTOR[=(*option*[,...])]

Dumps the specified ISO 9660 volume descriptors in a formatted manner. If **/NOFORMATTED** is specified, block mode format is used.

The descriptor options that you can specify are as follows:

Option	Description
BOOT:*n*	Searches for the *n*th occurrence of a Boot Record.
PVD:*n*	Searches for the *n*th occurrence of a Primary Volume Descriptor.
SVD:*n*	Searches for the *n*th occurrence of a Supplementary Volume Descriptor.
VPD:*n*	Searches for the *n*th occurrence of a Volume Partition Descriptor.
VDST:*n*	Searches for the *n*th occurrence of a Volume Descriptor Set Terminator.

If you specify only one option, you can omit the parentheses.

ISO 9660 descriptors are specified by their ordinal position from the start of the volume, defaulting to 1 if they are not specified. The ISO 9660 volume is sequentially searched from the beginning of the volume descriptor set sequence to the end to find the specified descriptor and output it in a formatted manner.

/DIRECTORY

Dumps data blocks of the specified file as formatted on-disk structures for Files-11 On-Disk Structure Level 1, 2, or 5 directory records, ISO 9660, or High Sierra directory records.

/EXACT

Use with the **/PAGE=SAVE** and **/SEARCH** qualifiers to specify a search string that must match the search string exactly and must be enclosed with quotation marks (" ").

If you specify the **/EXACT** qualifier without the **/SEARCH** qualifier, exact search mode is enabled when you set the search string with the Find (**E1**) key.

/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.

/HIGHLIGHT[=*keyword*]

Use with the **/PAGE=SAVE** and **/SEARCH** qualifiers to specify the type of highlighting you want when a search string is found. When a string is found, the entire line is highlighted. You can use the following keywords: BOLD, BLINK, REVERSE, and UNDERLINE. BOLD is the default highlighting.

/IDENTIFIER=*file-id*

Dumps the file selected by the file identification (FID) number from the specified volume. For further information, see the **/FILE_ID** qualifier of the **DIRECTORY** command.

/LONGWORD (default)

Formats the dump in longwords. The **/BYTE**, **/LONGWORD**, and **/WORD** qualifiers are mutually exclusive.

/MEDIA_FORMAT=*keyword*

Specifies the format in which a data structure is to be dumped. If you specify this qualifier, you must use one of the following keywords:

Keyword	Description
CDROM	Specifies ISO 9660 media format. This format is the default if you do not specify the **/MEDIA_FORMAT** qualifier.
CDROM_HS	Specifies High Sierra media format.

/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. The **/OUTPUT** and **/PRINTER** qualifiers are mutually exclusive.

/PAGE[=*keyword*]
/NOPAGE (default)

Controls the display of dump information on the screen.

You can use the following keywords with the **/PAGE** qualifier:

Keyword	Description
CLEAR_SCREEN	Clears the screen before each page is displayed.

Keyword	Description
SCROLL	Displays information one line at a time.
SAVE[=*n*]	Enables screen navigation of information, where *n* is the number of pages to store.

The **/PAGE=SAVE** qualifier allows you to navigate through screens of information. The **/PAGE=SAVE** qualifier stores up to 5 screens of up to 255 columns of information. When you use the **/PAGE=SAVE** qualifier, you can use the following keys to navigate through the information:

Key Sequence	Description
Up arrow key, **Ctrl/B**	Scroll up one line.
Down arrow key	Scroll down one line.
Left arrow key	Scroll left one column.
Right arrow key	Scroll right one column.
Find (**E1**)	Specify a string to find when the information is displayed.
Insert Here (**E2**)	Scroll right one half screen.
Remove (**E3**)	Scroll left one half screen.
Select (**E4**)	Toggle 80/132 column mode.
Prev Screen (**E5**)	Get the previous page of information.
Next Screen (**E6**), **Return**, **Enter**, **Space**	Get the next page of information.
F10, **Ctrl/Z**	Exit. Some utilities define these differently.
Help (**F15**)	Display utility help text.
Do (**F16**)	Toggle the display to oldest/newest page.
Ctrl/W	Refresh the display.

The **/PAGE** qualifier is not compatible with the **/OUTPUT** qualifier.

/PATH_TABLE

Dumps data blocks in ISO 9660 Path Table format.

/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. The asterisk (*) and the percent sign (%) wildcard characters are not allowed. The **/OUTPUT** and **/PRINTER** qualifiers are mutually exclusive.

/PROCESS

Attempts to generate a process dump.

/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.

Records are numbered beginning with 1.

Select a range of records to be dumped by specifying one of the following options:

Option	Description
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.

/SEARCH="*string*"

Use with the **/PAGE=SAVE** qualifier to specify a string that you want to find in the information being displayed. Quotation marks are required for the **/SEARCH** qualifier, if you include spaces in the text string.

You can also dynamically change the search string by pressing the Find key (**E1**) while the information is being displayed. Quotation marks are not required for a dynamic search.

/STYLE=*keyword*

Specifies the file name format for display purposes while performing a file dump.

The valid keywords for this qualifier are CONDENSED and EXPANDED. Descriptions are as follows:

Keyword	Description
CONDENSED (default)	Displays the file name representation of what is generated to fit into a 255-length character string. This file name may contain a DID or FID abbreviation in the file specification.
EXPANDED	Displays the file name representation of what is stored on disk. This file name does not contain any DID or FID abbreviations.

The keywords CONDENSED and EXPANDED are mutually exclusive. This qualifier specifies which file name format is displayed in the output header.

File errors are displayed with the CONDENSED file specification unless the EXPANDED keyword is specified.

See the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/] for more information.

/SYMLINK=*keyword*
/NOSYMLINK (default)

If an input file is a symbolic link, the file referred to by the symbolic link is the file that is dumped.

The **/SYMLINK** qualifier indicates that any input symbolic link is dumped.

The valid keywords for this qualifier are [NO]WILDCARD, [NO]ELLIPSIS, and [NO]TARGET. Descriptions are as follows:

Keyword	Description
NOWILDCARD	Indicates that symlinks are disabled during directory wildcard searches.
WILDCARD	Indicates that symlinks are enabled during wildcard searches.
NOELLIPSIS	Indicates that symlinks are matched for all wildcard fields except for ellipsis.
ELLIPSIS	Equivalent to WILDCARD (included for command symmetry).
TARGET	Indicates that if the named file is a symlink, then the symlink is followed to operate on the symlink target.
NOTARGET	Indicates that the command operates on the named file whether it is an ordinary file or a symlink.

If the file named in the **DUMP** command is a symlink, the command by default operates on the symlink target.

/VALIDATE_HEADER

Verifies **/DIRECTORY** records for Files-11.

/WIDTH=*n*

Formats the dump output into 80 or 132 columns by specifying *n* as either 80 or 132.

/WORD

Formats the dump in words. The **/BYTE**, **/LONGWORD**, and **/WORD** qualifiers are mutually exclusive.

/WRAP
/NOWRAP (default)

Use with the **/PAGE=SAVE** qualifier to limit the number of columns to the width of the screen and to wrap lines that extend beyond the width of the screen to the next line.

The **/NOWRAP** qualifier extends lines beyond the width of the screen and can be seen when you use the scrolling (left and right) features provided by the **/PAGE=SAVE** qualifier.

Examples

1.
```
$ DUMP TEST.DAT
Dump of file DISK0:[MOORE]TEST.DAT;1 on 14-DEC-2001 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.

2.
```
$ DUMP TEST.DAT/OCTAL/BYTE
Dump of file DISK0:[SCHELL]TEST.DAT;1 on 14-DEC-2001 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.

4.
```
$ DUMP/HEADER/BLOCK=COUNT=0 SYS$SYSTEM:DATASHARE.EXE

Dump of file SYS$SYSTEM:DATASHARE.EXE on  12-NOV-2001 16:06:46.75
File ID (16706,59,0)   End of file block 410 / Allocated 411

                              File Header

Header area
    Identification area offset:           40
    Map area offset:                      100
    Access control area offset:           255
    Reserved area offset:                 255
    Extension segment number:             0
    Structure level and version:          2, 1
    File identification:                  (16706,59,0)
    Extension file identification:        (0,0,0)
    VAX RMS attributes
        Record type:                      Fixed
        File organization:                Sequential
        Record attributes:                <none specified>
        Record size:                      512
        Highest block:                    411
        End of file block:                410
```

```
        End of file byte:                 414
        Bucket size:                      0
        Fixed control area size:          0
        Maximum record size:              512
        Default extension size:           0
        Global buffer count:              0
        Directory version limit:          0
    File characteristics:                 Contiguous best try
    Caching attribute:                    Writethrough
    Map area words in use:                3
    Access mode:                          0
    File owner UIC:                       [1,4]
    File protection:                      S:RWED, O:RWED, G:RE, W:
    Back link file identification:        (7149,80,0)
    Journal control flags:                <none specified>
    Active recovery units:                None
    Highest block written:                411
    Client attributes:                    None

Identification area
    File name:                            DATASHARE.EXE
    Revision number:                      1
    Creation date:                        12-AUG-2001 14:06:49.84
    Revision date:                        12-AUG-2001 14:06:53.20
    Expiration date:                      <none specified>
    Backup date:                          <none specified>

Map area
    Retrieval pointers
        Count:        411        LBN:    1297155

Checksum:                                 30710
```

In this example, the **DUMP** command dumps the file header of the specified file. Because this file is recorded on Files-11 ODS-2 9660 media, the file header is displayed in a Files-11 File Header format. Imbedded on the Files-11 Header is a VAX RMS attributes block.

5.
```
$ DUMP/HEADER/BLOCK=COUNT=0 DISK$GRIPS_2:[000000]AAREADME.TXT;
Dump of file DISK$GRIPS_2:[000000]AAREADME.TXT;1 on 15-DEC-2001
10:07:29.70

   File ID (4,6,0)   End of file block 29 / Allocated 29

                            ISO 9660 File Header

   Length of Directory Record:              48
   Extended Attribute Length:               1
   Location of Extent (LSB/MSB):            312/312
   Data Length of File Section (LSB/MSB):   14640/14640
   Recording Date and Time                  10-DEC-2001 16:22:30 GMT(0)
   File Flags                               RECORD, PROTECTION
   Interleave File Unit size:               0
   Interleave Gap size:                     0
   Volume Sequence # of extent (LSB/MSB):   1/1
   File Identifier Field Length:            14
   File Identifier:                         AAREADME.TXT;1
   System Use
5458542E 454D4441 45524141 0E010000 01000018 001E1610 100B5930 39000000
```

```
...90Y.............AAREADME.TXT 000000
                                                                  00313B
;1............................. 000020

Extended Attribute record
   Owner Identification (LSB/MSB):         7/7
   Group Identification (LSB/MSB):         246/246
   Access permission for classes of users S:R, O:R, G:RE, W:RE
   File Creation Date/Time:                5-OCT-2001 14:17:49.29 GMT(0)
   File Modification Date/Time:            6-NOV-2001 16:22:30.96 GMT(0)
   File Expiration Date/Time:              00-00-0000 00:00:00.00 GMT(0)
   File Effective Date/Time:               00-00-0000 00:00:00.00 GMT(0)
   Record Format                           Fixed
   Record Attributes                       CRLF
   Record Length (LSB/MSB):                80/80
   System Identifier:
   System Use
   Extended Attribute Version:             1
   Escape Sequence record length:          0
   Application Use Length (LSB/MSB):       0/0
   Application Use

   VAX RMS attributes
       Record type:                      Fixed
       File organization:                Sequential
       Record attributes:                Implied carriage control
       Record size:                      80
       Highest block:                    29
       End of file block:                29
       End of file byte:                 304
       Bucket size:                      0
       Fixed control area size:          0
       Maximum record size:              80
       Default extension size:           0
       Global buffer count:              0
       Directory version limit:          0
```

The **DUMP/HEADER** command dumps the file header of the specified file. Because this file is recorded on ISO 9660 media, the file header is displayed in the format of an ISO 9660 File Header and, since this file contains an optional ISO 9660 Extended Attribute Record (XAR), it is also displayed. Finally, as with all **DUMP/HEADER** requests, VAX RMS attributes are displayed.

EDIT/ACL

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 more information about the ACL Editor, see the relevant section in the *VSI OpenVMS System Management Utilities Reference Manual* [https://docs.vmssoftware.com/vsi-openvms-system-management-utilities-reference-manual-volume-i-a-l/#ACL_PART], the *VSI OpenVMS Guide to System Security* [https://docs.vmssoftware.com/vsi-openvms-guide-to-system-security/], or online help.

Format

EDIT/ACL *object-spec*

EDIT/EDT

EDIT/EDT — Invokes EDT, an interactive text editor. The **/EDT** qualifier is required. Information on EDT commands is available from within EDT by pressing **Ctrl/Z** and typing HELP at the EDT Command prompt. In addition to command help, you can also press PF2 for keypad help. For a description of EDT, including information about EDT commands and qualifiers, see the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/].

Format

EDIT/EDT *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.

The asterisk (*) and the percent sign (%) wildcard characters are not 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 *OpenVMS 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. The asterisk (*) and the percent sign (%) wildcard characters are not 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/EDT/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/EDT/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/EDT/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/EDT/NOJOURNAL MEMO.DAT
```

Once you have created a journal, enter the **EDT/RECOVER** command to execute the commands in the journal. The asterisk (*) and the percent sign (%) wildcard characters are not 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/EDT/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/EDT/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/EDT/NOOUTPUT MEMO.DAT
```

A system interruption does not prevent you from recreating 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.

The asterisk (*) and the percent sign (%) wildcard characters are not 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/EDT/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/EDT/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/EDT/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/EDT/OUTPUT=NEWFILE.TXT OLDFILE.TXT
     1       This is the first line of the file OLDFILE.TXT.
*
```

This 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/EDT/RECOVER OLDFILE.TXT
```

This 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

EDIT/FDL — Invokes the Edit/FDL (File Definition Language) utility, which creates and modifies FDL files. The **/FDL** qualifier is required. For more information about the File Definition Language utility, see the relevant sections in the *VSI OpenVMS Record Management Utilities Reference Manual* [https://docs.vmssoftware.com/vsi-openvms-record-management-utilities-reference-manual/#FDL_FACILITY] or online help.

Format

EDIT/FDL *filespec*

EDIT/SUM

EDIT/SUM — Invokes the SUMSLP utility, a batch-oriented editor, to update a single input file with multiple files of edit commands. For more information about the SUMSLP utility, see the *OpenVMS SUMSLP Utility Manual* or online help.

Format

EDIT/SUM *input-file*

EDIT/TECO

EDIT/TECO — Invokes the TECO interactive text editor.

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. The asterisk (*) and the percent sign (%) wildcard characters are not 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.

argument
See the **/EXECUTE** qualifier.

Description

The TECO editor creates or edits text files. For detailed information on the use of TECO, see the *Standard TECO Text Editor and Corrector for the VAX, PDP-11, PDP-10, and PDP-8* 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 *Standard TECO Text Editor and Corrector for the VAX, PDP-11, PDP-10, and PDP-8* manual.

To prevent TECO from using any startup command file, use the **/NOCOMMAND** qualifier as follows:

```
$ EDIT/TECO/NOCOMMAND MEMO.DAT
```

The asterisk (*) and the percent sign (%) wildcard characters are not 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 *Standard TECO Text Editor and Corrector for the VAX, PDP-11, PDP-10, and PDP-8* 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
```

The asterisk (*) and the percent sign (%) wildcard characters are not 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

EDIT/TPU — Invokes the DEC Text Processing utility (DECTPU). By default, this runs the Extensible Versatile Editor (EVE). DECTPU provides a structured programming language and other components for creating text editors and other applications. EVE is a general-purpose text editor that is the OpenVMS default editor. For more information about editing with EVE, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#EVE_CH] or online help.

Format

EDIT[/TPU] [*input-file*]

ENABLE AUTOSTART

ENABLE AUTOSTART — Enables the autostart feature on a node for all autostart queues managed by the specified queue manager. By default, this command uses the **/QUEUES** qualifier. For more information on autostart queues, see the relevant section in the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/#_6017QUEUES].

Format

ENABLE AUTOSTART[/QUEUES]

Description

Note

The command requires OPER (operator) privileges.

Enabling autostart for queues notifies the queue manager to automatically start all of its stopped active autostart queues on a node. It also notifies the queue manager to automatically start any of its autostart queues that fail over to the node. By default, the **ENABLE AUTOSTART** command affects the node from which it is entered. Specify the **/ON_NODE** qualifier to enable autostart on a different node.

By default, the command affects autostart queues managed by the default queue manager, SYS$QUEUE_MANAGER. Specify the **/NAME_OF_MANAGER** qualifier to disable autostart of a different queue manager's autostart queues on the node.

An autostart queue is active if it has been activated by the **/START** qualifier with the **INITIALIZE/QUEUE** command or by the **START/QUEUE** command and has not been stopped by the **STOP/QUEUE/NEXT** or **STOP/QUEUE/RESET** command.

When a node boots, autostart is disabled until you enter the **ENABLE AUTOSTART** command. Typically, you would add this command to your site-specific startup command procedure or your queue startup command procedure to start a node's autostart queues each time the node boots.

Qualifiers

/NAME_OF_MANAGER=*name*

Specifies the name of the queue manager controlling the autostart queues you want to enable. The qualifier allows the autostart feature to be used differently for different sets of queues.

If the **/NAME_OF_MANAGER** qualifier is omitted, the default queue manager name SYS$QUEUE_MANAGER is used.

For more information on multiple queue managers, see the relevant section in the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/#CREATE_ADD_QM].

/ON_NODE=*nodename*

Specifies a node in an OpenVMS Cluster system. Use this qualifier to enable autostart on a node other than the one from which you enter the command.

/QUEUES

Specifies that autostart is to be enabled for queues. This qualifier is used by default.

Examples

1.
```
$ INITIALIZE/QUEUE/BATCH/START-
_$ /AUTOSTART_ON=SATURN:: BATCH_1
$ ENABLE AUTOSTART/QUEUES
   .
   .
   .
$ DISABLE AUTOSTART/QUEUES
```

   In this example, the **INITIALIZE/QUEUE** command creates an autostart queue BATCH_1, capable of running on node SATURN. The **/START** qualifier activates the queue for autostart. The **ENABLE/AUTOSTART/QUEUES** command (executed on node SATURN) enables autostart on the node, causing the queue (and any other active autostart queues on the node) to begin executing jobs.

The **DISABLE AUTOSTART** command (executed on node SATURN) stops autostart queues on the node and prevents any queues from failing over to the node.

These commands only affect queues managed by the default queue manager SYS$QUEUE_MANAGER because the **/NAME_OF_MANAGER** qualifier is not specified.

Because BATCH_1 is set up to run only on one node, the queue cannot fail over to another node and therefore is stopped; however, the queue remains active for autostart and will be started when the **ENABLE AUTOSTART** command is entered for node SATURN. No **START/QUEUE** command is needed to restart BATCH_1 unless autostart of the queue is deactivated with the **STOP/QUEUE/NEXT** or **STOP/QUEUE/RESET** command.

2.
```
$ INITIALIZE/QUEUE/BATCH/START-
_$ /AUTOSTART_ON=(NEPTUN::,SATURN::) BATCH_1
$ ENABLE AUTOSTART/QUEUES/ON_NODE=NEPTUN
$ ENABLE AUTOSTART/QUEUES/ON_NODE=SATURN
  .
  .
  .
$ STOP/QUEUES/ON_NODE=NEPTUN
```

In this example, the **INITIALIZE/QUEUE** command creates an autostart queue BATCH_1. The **/START** qualifier activates the queue for autostart.

The first **ENABLE AUTOSTART/QUEUES** command causes the queue to begin executing on node NEPTUN. The second **ENABLE AUTOSTART/QUEUES** command enables autostart on node SATURN to start all stopped active autostart queues on that node and to start any autostart queues that might fail over to that node.

Later, suppose node NEPTUN must be removed from the OpenVMS Cluster system. The **STOP/QUEUES/ON_NODE** command stops all queues on node NEPTUN, and causes the autostart queue BATCH_1 to fail over to node SATURN. Because the queue is active for autostart, and because autostart has been enabled on node SATURN, the queue is automatically started on that node.

This command only affects queues managed by the default queue manager SYS$QUEUE_MANAGER because the **/NAME_OF_MANAGER** qualifier is not specified.

ENCRYPT

ENCRYPT — Encrypts files by default with the Data Encryption Standard (DES) algorithm in Cipher Block Chaining (CBC) mode unless otherwise specified with the **/KEY_ALGORITHM** and **/DATA ALGORITHM** qualifiers. Before you enter this command, create a key with the **ENCRYPT/CREATE_KEY** command. The key specified must match the algorithm (DES or AES).

Format

ENCRYPT*input-file key-name* [*qualifiers*]

Parameters

input-file

File names of the files to encrypt. If you use wildcard characters, do not include directory files or files with bad blocks.

key-name

Key name previously stored in the key storage table with the **ENCRYPT/CREATE_KEY** command.

Qualifiers

/BACKUP[=*time*]

Selects files according to the dates of their most recent backup.

This qualifier is relevant only when used with the **/BEFORE** or the **/SINCE** qualifier. In addition, do not use **/BACKUP** with **/EXPIRED** or **/MODIFIED**.

If you omit *time*, TODAY is used. For more information on time specifications, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT].

/BEFORE[=*time*]

Selects files that have a creation time before the time you specify.

If you omit *time*, TODAY is used. For more information on time specifications, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT].

/BY_OWNER[=*uic*]
/NOBY_OWNER

Selects files with the owner UIC you specify.

If you omit *uic*, the UIC of the current process is used. For more information on specifying UIC format, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#UIC_FORMAT_DIR].

/COMPRESS
/NOCOMPRESS

Optional. Default: **/NOCOMPRESS**.

Controls whether or not data compression occurs before a file is encrypted.

/CONFIRM
/NOCONFIRM

Controls whether or not a confirmation request is displayed before each encryption, as follows:

Response	**Meaning**
YES	Encrypts the file
NO or **Return**	Does not encrypt the file (default)
QUIT or **Ctrl/Z**	Does not encrypt the file or any subsequent files
ALL	Encrypts the file plus all subsequent files

/DATA_ALGORITHM=

1. DESCBC (default)
2. AES*mmmkkk*

Where *mmm* is the mode CBC, ECB, CFB, or OFB; and *kkk* is 128, 192, or 256 bits. Cipher Block Chaining (CBC) and Electronic Code Book (ECB) are 16-byte block modes, meaning blocks are padded to 16 bytes if necessary during encryption. The padding is removed during decryption. Cipher Feedback (CFB) and Output Feedback (OFB) are 8-bit character stream mode emulation, useful in data communications and where no padding is required.

Note that **/DATA_ALGORITM=AES** is a shortcut for specifying AESCBC128.

The data algorithm is used with the randomly generated key to perform encryption of the file's data. When specifying an AES algorithm, specify both **/KEY** and **/DATA=AES*mmmkkk*** qualifiers and use an AES created key.

/DELETE
/NODELETE

Controls whether or not the input files are deleted after the encryption operation is complete and the output file is written and closed. By default, the input file is not deleted.

/ERASE
/NOERASE

Controls whether or not the input files are erased with the data security pattern before being deleted. By default, the location in which the data was stored is not overwritten with the data security pattern. The **/ERASE** qualifier must be used with **/DELETE**.

/EXCLUDE=*file-spec*
/NOEXCLUDE

Excludes the specified files from the encryption operation. You can use wildcard characters. You do not need to enter an entire file specification. Any field that you omit defaults to the input file specification.

Because directory files are never encrypted, you need not specify them.

/EXPIRED[=*time*]

Selects files according to the dates on which they expire.

This qualifier is relevant only when used with the **/BEFORE** or the **/SINCE** qualifier. In addition, do not use **/EXPIRED** with **/BACKUP** or **/MODIFIED**.

If you omit *time*, TODAY is used. For more information on time specifications, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT].

/KEY_ALGORITHM=

1. DESCBC (default)
2. AES*mmmkkk*

Where *mmm* is the mode CBC, ECB, CFB, or OFB; and *kkk* is 128, 192, or 256 bits. Note that **/KEY_ALGORITHM=AES** is a shortcut for specifying AESCBC128.

The command uses this key algorithm with the key you supply to encrypt the randomly generated data encryption key and the initialization vector stored within the file.

When specifying an AES algorithm, specify both **/KEY** and **/DATA** qualifiers and use an AES created key.

/MODIFIED[=*time*]

Selects files according to the dates on which they were last modified.

This qualifier is relevant only when used with the **/BEFORE** or the **/SINCE** qualifier. In addition, do not use **/MODIFIED** with **/BACKUP** or **/EXPIRED**.

If you omit *time*, TODAY is used. For more information on time specifications, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT].

/OUTPUT=*file-spec*

Alternate output file name for the encryption operation. By default, each input file encrypted is written to a separate output file that is one version higher than the highest version of the input file. When using the **/OUTPUT** qualifier, specify the parts of the file specification different from the defaults. You do not need to provide an entire file specification. Any field that you omit defaults to the input file specification.

/SHOW=*keyword-list*

Controls whether or not the following information about the encryption operation is displayed on SYS$COMMAND:

Keyword	Description
FILES	Displays input and output file names on SYS$COMMAND
STATISTICS	Displays the encryption stream statistics: • Bytes processed • Internal records processed • CPU time consumed within the encryption algorithm

/SINCE[=*time*]

Selects files that have a creation date before the time you specify.

If you omit *time*, TODAY is used. For more information on time specifications, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT].

/STATISTICS

Similar to **/SHOW**, except that **/STATISTICS** lists both files and statistics, whereas **/SHOW** can be customized to list only one or the other.

/VERSION

Displays the version number of the Encryption for OpenVMS software running on your system.

Examples

1. `$ ENCRYPT TROY MYKEY`

   Encrypts the file `TROY` using the key `MYKEY`.

2. `$ ENCRYPT NEWFILE.TXT MONET/KEY_ALGORITHM=AESCBC128/ DATA_ALGORITHM=AESCBC128`

   Encrypts the file `NEWFILE.TXT` with the AES key, `MONET`, using the algorithm `AESCBC128`. A new version, NEWFILE.TXT;n+1, of the original file (now encrypted) is created. Use the **`/OUTPUT=filename`** qualifier to preserve the original file name, renaming the encrypted output file.

ENCRYPT/AUTHENTICATE

ENCRYPT/AUTHENTICATE — Associates a DES algorithm Message Authenticate Code (MAC) value with one or more files and checks for any modification of either plain text or cipher text files. Use the additional **`/UPDATE`** qualifier to store each file's MAC in the databases. Use only the **`/AUTHENTICATE`** qualifier to subsequently test the integrity of the file's data and security attributes. You must create a DES key prior to updating or checking an existing MAC. The AES algorithm is not supported for file MAC operations.

Format

`ENCRYPT/AUTHENTICATE`*`file-spec key-name`* `[`*`qualifiers`*`]`

Parameters

`file-spec`

File names of the files to authenticate. Behavior can be modified with the **`/MULTIPLE_FILES`** qualifier.

`key-name`

Key name previously stored in the key storage table with the **`ENCRYPT/CREATE_KEY`** command.

Qualifiers

/BACKUP[=*time*]

Selects files according to the dates of their most recent backup.

This qualifier is relevant only when used with the **`/BEFORE`** or the **`/SINCE`** qualifier. In addition, do not use **`/BACKUP`** with **`/EXPIRED`** or **`/MODIFIED`**.

If you omit *time*, TODAY is used. For more information on time specifications, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT].

/BEFORE=*time*

Selects files that have a creation time before the time you specify.

If you omit *time*, TODAY is used. For more information on time specifications, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT].

/BY_OWNER[=*uic*]
/NOBY_OWNER

Selects files with the owner UIC you specify.

If you omit *uic*, the UIC of the current process is used. For more information on specifying UIC format, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#UIC_FORMAT_DIR].

/CONFIRM
/NOCONFIRM

Controls whether or not a confirmation request is displayed before each authentication, as follows:

Response	**Meaning**
YES	Authenticates the file
NO or **Return**	Does not authenticate the file (default)
QUIT or **Ctrl/Z**	Does not authenticate the file or any subsequent files
ALL	Encrypts the file plus all subsequent files

/DATABASE=*file-spec*
/NODATABASE

File name of the file in which to store binary MAC values.

Generates a MAC using the file contents. If you do not specify a file name, the file name SYS$LOGIN:ENCRYPT$MAC.DAT is used.

/EXCLUDE=*file-spec*
/NOEXCLUDE

Excludes the specified files from the authentication operation. You can use wildcard characters. You do not need to enter an entire file specification. Any field that you omit defaults to the input file specification.

Because directory files are never encrypted, you need not specify them.

/EXPIRED[=*time*]

Selects files according to the dates on which they expire.

This qualifier is relevant only when used with the **/BEFORE** or the **/SINCE** qualifier. In addition, do not use **/EXPIRED** with **/BACKUP** or **/MODIFIED**.

If you omit *time*, TODAY is used. For more information on time specifications, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT].

/LOG

Displays the results of the authentication operation.

/MODIFIED[=*time*]

Selects files according to the dates on which they were last modified.

This qualifier is relevant only when used with the **/BEFORE** or the **/SINCE** qualifier. In addition, do not use **/MODIFIED** with **/BACKUP** or **/EXPIRED**.

If you omit *time*, TODAY is used. For more information on time specifications, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT].

/MULTIPLE_FILES

Indicates that the *file-spec* parameter contains a list of file names to be checked. The *file-spec* file is opened and each record is read and treated as a *file-spec*.

/OUTPUT=*file-spec*
/NOOUTPUT

File name of the file in which to store readable MAC values. These MAC values represent both the file contents as well as the security settings. If you do not specify a file name, the default file name SYS$LOGIN:ENCRYPT$MAC.LIS is used.

/SECURITY=*file-spec*
/NOSECURITY

File name of the file in which to store binary MAC values. If you do not specify a file name, the default file name ENCRYPT$SEC.DAT is used.

Generates a MAC using the file's security settings: owner, protection settings, and optional ACL.

/SINCE[=*time*]

Selects files that have a creation time before the time you specify.

If you omit *time*, TODAY is used. For more information on time specifications, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT].

/UPDATE
/NOUPDATE

Associates new MAC values with one or more files.

Example

1.
```
$ ENCRYPT/AUTHENTICATE NEWFILE HAMLET/CONFIRM
```

   Associates a MAC with the file `NEWFILE` using the key `HAMLET`. This command also displays a confirmation request before each authentication.

2.
```
$ ENCRYPT/AUTHENTICATE/UPDATE *.* MYKEY
%ENCRYPT-NEWDB, new authentication code database has been created
```

```
%ENCRYPT-NEWSECDB, new authentication security settings database has
 been created
%ENCRYPT-I-SUMMARY1, Summary:  Files successfully authenticated: 0
%ENCRYPT-I-SUMMARY2,     Files failing authentication: 0
%ENCRYPT-I-SUMMARY3,     Files not in database: 73
%ENCRYPT-I-SECSUMM1, Summary:  Security settings authenticated: 0
%ENCRYPT-I-SECSUMM2,     Security settings failing authentication: 0
%ENCRYPT-I-SECSUMM3,     Security settings not in database: 73
```

This example creates a MAC for each file in the current directory using the key named MYKEY, storing them in the two databases: SYS$LOGIN:ENCRYPT$MAC.DAT and ENCRYPT$SEC_MAC.DAT.

3.
```
$ ENCRYPT/AUTHENTICATE *.* MYKEY
%ENCRYPT-I-NOUPDATE, database will not be updated with new
 authentication codes
%ENCRYPT-I-SUMMARY1, Summary:   Files successfully authenticated: 73
%ENCRYPT-I-SUMMARY2,     Files failing authentication: 0
%ENCRYPT-I-SUMMARY3,     Files not in database: 0
%ENCRYPT-I-SECSUMM1, Summary:   Security settings authenticated: 73
%ENCRYPT-I-SECSUMM2,     Security settings failing authentication: 0
%ENCRYPT-I-SECSUMM3,     Security settings not in database: 0
```

This example authenticates the same files as in Example 2 by creating a new MAC and comparing that with those in each database, testing file data integrity and security attributes as indicated in the summary.

ENCRYPT/CREATE_KEY

ENCRYPT/CREATE_KEY — Creates a key definition name and value to be used for encrypting and decrypting files. The key is a string that represents the name under which its value is encrypted and stored in the key storage table; a logical name table. A DES key is created in the PROCESS logical name table by default unless the **/AES** qualifier is specified. Note that AES requires longer key-length values than the 8-byte DES keys. AES requires a minimum of 16, 24, or 32 bytes depending on the algorithm/key size specified for encryption or decryption.

Format

ENCRYPT/CREATE_KEY*key-name key-value* [*qualifiers*]

Parameters

key-name

Name under which the encryption key will be stored in the key storage table. Specify a character string according to the following conventions:

- 1 to 243 alphanumeric characters
- Dollar signs and underscores are valid.
- Not case sensitive

Use a name that has meaning to you, to help you remember it.

Note

Key names beginning with ENCRYPT$ are reserved for OpenVMS.

key-value

String representing the value of the encryption key. Specify either ASCII text or a hexadecimal constant, as follows:

- ASCII text string (default)
  - Minimum length: 8 (DES) 16, 24, or 32 (AES -- 128, 192, and 256 bits respectively).
  - Maximum length: approximately 240 characters.
  - The string is not case sensitive for DES keys.
  - If you use characters other than alphanumeric characters, for example, blank spaces, enclose the string in quotation marks (" ").
- Hexadecimal constant
  - Use the **/HEXADECIMAL** qualifier.
  - Valid characters: 0 to 9, A to F (ASCII coded HEX nibbles).
  - Minimum length: 16 characters -- DES -- 32, 48, or 64 (AES -- 128, 192, and 256 bits respectively).
  - Do **not** enclose the value in quotation marks.

Qualifies

/AES

Designates that an AES key is to be created, which is encrypted with the AESCBC128 encryption routine.

/GROUP

Enters the key definition in the group key storage table.

/HEXADECIMAL
/NOHEXADECIMAL

Specifies that the value for the key is a hexadecimal number. Default: key values are interpreted as ASCII text characters (see the description of the *key-value* parameter).

/JOB

Enters the key definition in the job key storage table.

/LOG

Verifies successful creation of the key.

/PROCESS

Enters the key definition in the process key storage table.

/SYSTEM

Enters the key definition in the system key storage table.

Examples

1. ```
   $ ENCRYPT/CREATE_KEY HAMLET
   _ Key value: "And you yourself shall keep the key of it"
   ```

   This example defines a DES key named HAMLET with the character string value.

   ```
   "And you yourself shall keep the key of it"
   ```

2. ```
   $ ENCRYPT/CREATE_KEY/HEXADECIMAL ARCANE 2F4A98F46BBC11DC
   ```

   This example defines a DES key named ARCANE with hexadecimal value of 2F4A98F46BBC11DC.

3. ```
   $ ENCRYPT/CREATE_KEY MYKEY "The 16 char. key" /LOG/AES
   ```

   This example defines an AES key named MYKEY with the minimum 16-character string value "The 16 char. key" that is required for AES*xxx*128, logging its successful creation. The key is encrypted with AES prior to storage in the PROCESS (default) logical name table.

4. ```
   $ SHOW LOGICAL ENC* /TABLE=ENCRYPT$KEY_STORE

   LNM$PROCESS_TABLE

     "ENCRYPT$KEY$MYKEY" =  ".0S%M.....SB.}0L..Z"
           = "AES"

   LNM$JOB_8210B400

   LNM$GROUP_000001

   ENCRYPT$SYSTEM
   ```

   This example shows that key names are prepended with ENCRYPTKEY, as in the named key ENCRYPTKEYMYKEY.

ENCRYPT/REMOVE_KEY

ENCRYPT/REMOVE_KEY — Deletes a key definition from a key storage table. The PROCESS logical name table is the default unless otherwise specified.

Format

ENCRYPT/REMOVE_KEY*key-name* [*qualifiers*]

Parameters

key-name

Key name previously stored in the key storage table with the **ENCRYPT/CREATE_KEY** command.

Qualifiers

/AES

Designates that an AES key is to be deleted. Specifying a unique key name and table is sufficient for deletion, making the **/AES** qualifier unnecessary but included for clarification.

/GROUP

Deletes the key definition from the group key storage table.

/JOB

Deletes the key definition from the job key storage table.

/PROCESS

Deletes the key definition from the process key storage table.

/SYSTEM

Deletes the key definition from the system key storage table.

Example

```
$ ENCRYPT /REMOVE_KEY MYKey /AES
```

This command removes or deletes the AES key, MYKEY.

ENDSUBROUTINE

ENDSUBROUTINE — Defines the end of a subroutine in a command procedure. For more information about the command, see the description of the **CALL** command or online help.

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:

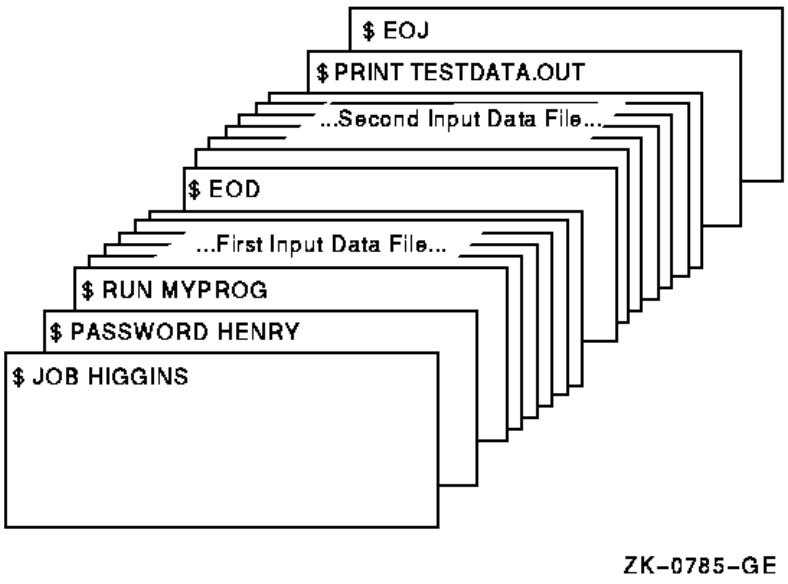
- Terminates input data lines that begin with dollar signs ($).The **DECK** command indicates that the following lines begin with dollar signs and should be interpreted as data, not as commands; the **EOD** command indicates the end of the data lines.
- Terminates an input file if multiple input files 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. 
   ```

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

EOJ — Marks the end of a batch job submitted through a card reader.

Synopsis

```
$ EOJ
```

Parameters

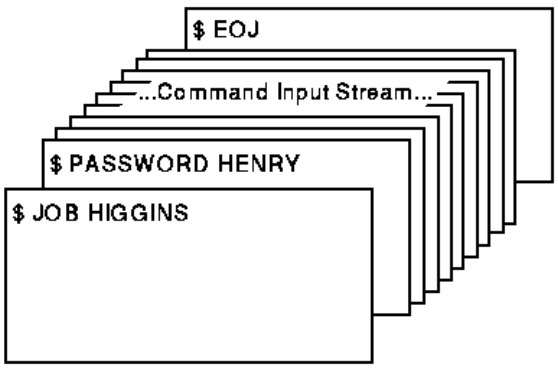
None.

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



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

EXAMINE — Displays the contents of virtual memory.

Synopsis

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

Note

The command requires user-mode read (R) access to the virtual memory location whose contents you want to examine.

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, one 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, one 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, one longword at a time.

/OCTAL

Displays the contents of the specified location in octal format.

/WORD

Displays data at the specified location, one 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

EXCHANGE — Invokes the Exchange utility (EXCHANGE), which manipulates mass storage volumes that are written in formats other than those normally recognized by the operating system. For more information about EXCHANGE, see the *OpenVMS Exchange Utility Manual* or online help.

Format

EXCHANGE[subcommand] [filespec] [filespec]

Description

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**.

EXCHANGE/NETWORK

EXCHANGE/NETWORK — Enables the operating system to transfer files to or from operating systems that do not support OpenVMS file organizations. The transfer occurs over a DECnet network communications link that connects OpenVMS systems and non OpenVMS operating system nodes.

Format

EXCHANGE/NETWORK*input-filespec*[,...] *output-filespec*

Parameters

input-filespec[,...]

Specifies the name of an existing file to be transferred. The asterisk (*) and the percent sign (%) 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 asterisk (*) 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 OpenVMS target node `MYPC`.

A more complete explanation of the asterisk (*) and the percent sign (%) wildcard characters and version numbers follows in the Description section.

Description

Using DECnet services, the **EXCHANGE/NETWORK** command can perform any of the following tasks:

- Transfer files between an OpenVMS node and a non OpenVMS system node.
- Transfer a group of input files to a group of output files.
- Transfer files between two non OpenVMS nodes, provided those nodes share DECnet connections with the OpenVMS node that issues the **EXCHANGE/NETWORK** command.

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 OpenVMS Record Management Services (RMS) facility provides the operating system access to records in OpenVMS RMS files. To transfer OpenVMS RMS files between two nodes where both nodes are OpenVMS 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 OpenVMS nodes and non OpenVMS 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 OpenVMS nodes and non OpenVMS nodes connected to the same DECnet network. If the non OpenVMS system does not support OpenVMS file organizations, the **EXCHANGE/NETWORK** command can modify or discard file and record attributes during the transfer. However, if the target system is an OpenVMS 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.

OpenVMS File and Record Attributes

All RMS files in the OpenVMS 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 OpenVMS RMS facility.

Non OpenVMS operating systems that do not support OpenVMS file organizations have no means of storing file and record attributes with their files. Transferring an OpenVMS file to a non OpenVMS 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 OpenVMS system.

Transferring Files to OpenVMS Nodes

When you transfer files to an OpenVMS system from a non OpenVMS 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. Alternatively, if transferring a CDA document, enter the following command after the **EXCHANGE/NETWORK** command:

```
$ SET FILE/SEMANTICS=[ddif,dtif] document-name.doc
```

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 OpenVMS system to an OpenVMS 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 relevant sections in the *VSI OpenVMS Record Management Utilities Reference Manual* [https://docs.vmssoftware.com/vsi-openvms-record-management-utilities-reference-manual/#FDL_FACILITY].

Transferring Files to Non OpenVMS Nodes

The **EXCHANGE/NETWORK** command discards file and record attributes associated with an OpenVMS file during a transfer to a non OpenVMS system that does not support OpenVMS 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 an OpenVMS node, use the AUTOMATIC transfer mode option.

Wildcard Characters

The asterisk (*) and the percent sign (%) wildcard characters are permitted in the file specifications and follow the behavior typical of other OpenVMS system commands with respect to the OpenVMS node.

When more than one input file is specified, but the asterisk (*) or the percent sign (%) wildcard characters 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 an OpenVMS 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 the asterisk (*) and the percent sign (%) 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 an OpenVMS 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 relevant section in the *VSI OpenVMS Guide to System Security* [https://docs.vmssoftware.com/vsi-openvms-guide-to-system-security/#ACL_DETAILS].

On OpenVMS 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 absolute time, as a combination of absolute and delta times, or as one of the following keywords: BOOT, LOGIN, 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 relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT] or the online help topic Date.

/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 *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#UIC_FORMAT_DIR].

/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 **Return**. 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. The asterisk (*) and the percent sign (%) wildcard characters 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 relevant sections in the *VSI OpenVMS Record Management Utilities Reference Manual* [https://docs.vmssoftware.com/vsi-openvms-record-management-utilities-reference-manual/#FDL_FACILITY].

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 on or after the specified time. You can specify time as absolute time, as a combination of absolute and delta times, or as one of the following keywords: BOOT, JOB_LOGIN, LOGIN, 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 relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT] or the online help topic Date.

/STYLE=*keyword*

Specifies the file name format for display purposes.

The valid keywords for this qualifier are CONDENSED and EXPANDED. Descriptions are as follows:

Keyword	Description
CONDENSED (default)	Displays the file name representation of what is generated to fit into a 255-length character string. This file name may contain a DID or FID abbreviation in the file specification.
EXPANDED	Displays the file name representation of what is stored on disk. This file name does not contain any DID or FID abbreviations.

The keywords CONDENSED and EXPANDED are mutually exclusive. This qualifier specifies which file name format is displayed in the output message, along with the confirmation if requested.

File errors are displayed with the CONDENSED file specification unless the EXPANDED keyword is specified.

See the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/] for more information.

/SYMLINK=*keyword*

The valid keywords for this qualifier are [NO]WILDCARD and [NO]ELLIPSIS. Descriptions are as follows:

Keyword	Description
WILDCARD	Indicates that symlinks are enabled during wildcard searches.
NOWILDCARD	Indicates that symlinks are disabled during directory wildcard searches.

Keyword	Description
ELLIPSIS	Equivalent to WILDCARD (included for command symmetry).
NOELLIPSIS	Indicates that symlinks are matched for all wildcard fields except for ellipsis.

If the file named in the **EXCHANGE/NETWORK** command is a symlink, the command operates on the symlink target.

/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	Description
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	Description
CARRIAGE_CONTROL	Any carriage control information in the input file is interpreted, expanded into actual characters, and 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 KUDOS::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 OpenVMS node KUDOS. 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.

2.
```
$ EXCHANGE/NETWORK/TRANSFER_MODE=BLOCK -
_$ KUDOS::FOREIGN_SYS.DAT VMS_FILE.DAT
```

   In this example, the **EXCHANGE/NETWORK** command transfers the file FOREIGN_SYS.DAT from the non OpenVMS node KUDOS to the file VMS_FILE.DAT in the current default device and directory. Block I/O is specified for the transfer mode.

3.
```
$ EXCHANGE/NETWORK/FDL=VMS_FILE_DEFINITION.FDL -
_$ KUDOS::REMOTE_FILE.TXT  VMS_FILE.DAT
```

   In this example, the **EXCHANGE/NETWORK** command transfers the file REMOTE_FILE.TXT on node KUDOS 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. Because the qualifier **/FDL** is specified and the **/TRANSFER_MODE** qualifier is omitted, the transfer mode uses block I/O, by default.

   For more information about creating FDL files, see the relevant section in the *VSI OpenVMS Record Management Utilities Reference Manual* [https://docs.vmssoftware.com/vsi-openvms-record-management-utilities-reference-manual/#CREATE_FDL_UTILITY].

4.
```
$ EXCHANGE/NETWORK -
_$ /TRANSFER_MODE=CONVERT=(CARRIAGE_CONTROL,COUNTED, -
_$ RECORD_SEPARATOR=CRLF,FIXED_CONTROL) -
_$ PRINT_FILE.TXT  KUDOS::*
```

   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 OpenVMS node KUDOS. 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 inserted before the data or 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 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

EXIT — Terminates processing of a command procedure or subroutine and returns control to the calling 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 non-interactive 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 non-interactive 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

Value	Severity
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

FONT — Converts an ASCII bitmap distribution format (BDF) into binary portable compiled format (PCF) on Alpha systems. The DECwindows server uses a PCF or 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, see the OpenVMS DECwindows programming documentation or online help.

Format

FONT ***filespec***

GOSUB

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

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

HELP — The **HELP** command invokes the Help facility to display information about use of the system, including formats and explanations of commands, parameters, qualifiers, and system messages.

Format

HELP [*topic*[*subtopic*...]]

Parameter

***topic*[*subtopic*...]**

Specifies the topics or topic and subtopics on which you want information from a help library.

Description

In response to the Topic? prompt, you can:

- Type the name of the command or topic for which you need help.
- Type **INSTRUCTIONS** for more detailed instructions on how to use **HELP**.
- Type **HINTS** if you are not sure of the name of the command or topic for which you need help.

- Type **/MESSAGE** for help with the **HELP/MESSAGE** utility.
- Type a question mark (?) to redisplay the most recently requested text.
- Press **Return** one or more times to exit from **HELP**.

You can abbreviate any topic name, although ambiguous abbreviations result in all matches being displayed.

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 previous levels – 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.

To use the Help facility on OpenVMS 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 **Return**. You can now ask for information on another topic when the Help facility displays the Topic? prompt. Press **Return** to exit the Help facility and return to DCL command level.

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, **HELPCOPY …** 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.

The asterisk (*) and the percent sign (%) wildcard characters are allowed in the keyword.

Qualifiers

/EXACT

Use with the **/PAGE=SAVE** and **/SEARCH** qualifiers to specify a search string that must match the search string exactly and must be enclosed with quotation marks (" ").

If you specify the **/EXACT** qualifier without the **/SEARCH** qualifier, exact search mode is enabled when you set the search string with the Find (**E1**) key.

/HIGHLIGHT[=*keyword*]

Use with the **/PAGE=SAVE** and **/SEARCH** qualifiers to specify the type of highlighting you want when a search string is found. When a string is found, the entire line is highlighted. You can use the following keywords: BOLD, BLINK, REVERSE, and UNDERLINE. BOLD is the default highlighting.

/INSTRUCTIONS (default)
/NOINSTRUCTIONS

Displays an explanation of the **HELP** command along with the list of topics (I 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.

/MESSAGE

Displays descriptions of system messages. See the **HELP/MESSAGE** command.

/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. The asterisk (*) and the percent sign (%) wildcard characters are not allowed.

If you enter the **/NOOUTPUT** qualifier, output is suppressed.

/PAGE[=*keyword*]
/NOPAGE (default)

Controls the display of information on the screen.

You can use the following keywords with the **/PAGE** qualifier:

Keyword	Description
CLEAR_SCREEN	Clears the screen before each page is displayed.
SCROLL	Displays information one line at a time.
SAVE[= *n*]	Enables screen navigation of information, where *n* is the number of pages to store.

The **/PAGE=SAVE** qualifier allows you to navigate through screens of information. The **/PAGE=SAVE** qualifier stores up to 5 screens of up to 255 columns of information. When you use the **/PAGE=SAVE** qualifier, you can use the following keys to navigate through the information:

Key Sequence	Description
Up arrow key, **Ctrl/B**	Scroll up one line.
Down arrow key	Scroll down one line.
Left arrow key	Scroll left one column.
Right arrow key	Scroll right one column.
Find (**E1**)	Specify a string to find when the information is displayed.
Insert Here (**E2**)	Scroll right one half screen.
Remove (**E3**)	Scroll left one half screen.
Select (**E4**)	Toggle 80/132 column mode.
Prev Screen (**E5**)	Get the previous page of information.
Next Screen (**E6**), **Return**, **Enter**, **Space**	Get the next page of information.
F10, **Ctrl/Z**	Exit. Some utilities define these differently.
Help (**F15**)	Display utility help text.
Do (**F16**)	Toggle the display to oldest/newest page.
Ctrl/W	Refresh the display.

The **/PAGE** qualifier is not compatible with the **/OUTPUT** qualifier.

/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.

/SEARCH="*string*"

Use with the **/PAGE=SAVE** qualifier to specify a string that you want to find in the information being displayed. Quotation marks are required for the **/SEARCH** qualifier, if you include spaces in the text string.

You can also dynamically change the search string by pressing the Find key (**E1**) while the information is being displayed. Quotation marks are not required for a dynamic search.

/USERLIBRARY=(*level*[,...])
/NOUSERLIBRARY

Names the levels of search for information in auxiliary libraries. The levels are as follows:

Level	Description
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.

/WRAP
/NOWRAP (default)

Use with the **/PAGE=SAVE** qualifier to limit the number of columns to the width of the screen and to wrap lines that extend beyond the width of the screen to the next line.

The **/NOWRAP** qualifier extends lines beyond the width of the screen and can be seen when you use the scrolling (left and right) features provided by the **/PAGE=SAVE** qualifier.

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 **Return** 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 **@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 OpenVMS 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.

HELP/MESSAGE

HELP/MESSAGE — Displays descriptions of system messages.

Format

HELP/MESSAGE[*/qualifier* [...]] [*search-string*]

Parameter

search-string

Specifies a message identifier or one or more words from a message's text. By default, **HELP/MESSAGE** displays a description of the message produced by the last executed command (that is, the message corresponding to the value currently stored in the CLI symbol $STATUS).

The Help Message utility (MSGHLP) operates on the search string using the following conventions:

- Words containing fewer than three alphanumeric characters are ignored.
- Words can be specified in any order.

  You can minimize search time by specifying the most unusual word first.
- Non-alphanumeric characters are ignored in the search. Exceptions are the percent sign (%) and hyphen (-) when they prefix a message; therefore, you can paste a full message into the search string, provided you include these special characters and delete any variables (such as file names) that were inserted into the message.

  If Help Message fails to find a pasted message in the database, submit the command again and omit the leading special character, facility, and severity. Some common messages are documented as "shared" messages rather than facility-specific messages.
- Help Message matches all words that begin with the characters specified in the search string. Use **/WORD_MATCH=WHOLE_WORD** to specify that only whole words be matched.

Description

The Help Message utility accesses message descriptions in a text file. This text file is derived from the latest *OpenVMS system messages documentation* and, optionally, from other source files, including

user-supplied message documentation. By default, Help Message provides information on how the last executed command completed.

You can extract all messages produced by one or more specified facilities. By directing this output to a file, you can create and print your own customized message documentation.

For full details about adding comments or messages to the Help Message database, see the *OpenVMS System Messages: Companion Guide for Help Message Users*.

Qualifiers

/BRIEF

Outputs the message text only.

/DELETE=*filename*.MSGHLP

Deletes all messages contained in the specified .MSGHLP file from whichever of the following files is found first:

- A .MSGHLP$DATA file specified with the **/LIBRARY** qualifier
- The first .MSGHLP$DATA file in a search path specified by the **/LIBRARY** qualifier
- The first .MSGHLP$DATA file in the default search path (defined by logical name MSGHLP$LIBRARY)
- SYS$HELP:MSGHLP$LIBRARY.MSGHLP$DATA (the default .MSGHLP$DATA file)

You must have write access to .MSGHLP$DATA files supplied by VSI to delete messages from the database.

Note

If you create a .MSGHLP file by specifying a search string, check the output .MSGHLP file to be sure the search did not pick up any unexpected messages that you do not want to delete from the database. Edit any such messages out of the .MSGHLP file before you perform the delete operation.

/EXTRACT=*filename*.MSGHLP

Extracts messages from the database and generates a .MSGHLP file that can be edited, if desired, and used as input for **/INSERT** and **/DELETE** operations. **/EXTRACT** retrieves data from a .MSGHLP$DATA file or logical search path specified by **/LIBRARY** or, by default, from files in the search path defined by logical name MSGHLP$LIBRARY. When **/EXTRACT** is not specified, Help Message produces output in standard text format by default (see the **/OUTPUT** qualifier).

/FACILITY=?
/FACILITY=(*facility-name* [,...])
/FACILITY=ALL

Specifies which facilities in the database are to be searched for a match.

Enter **/FACILITY=?** to output a list of all facilities in the default database or in a database specified by **/LIBRARY**.

To narrow your search, specify one or more facility names with **/FACILITY**. Multiple facilities must be enclosed in parentheses and be separated by commas. Help Message then outputs only matching messages produced by the specified facility or facilities.

Specify **/FACILITY=ALL** to output messages for all facilities in the database. **/FACILITY=ALL** is the default unless another facility is implied; for example, specifying **/STATUS** or defaulting to the value of the CLI symbol $STATUS automatically identifies a specific facility. Similarly, cutting and pasting a message that includes a facility name invalidates use of the **/FACILITY** qualifier.

See the *OpenVMS System Messages: Companion Guide for Help Message Users* for more details about using the **/FACILITY** qualifier.

/FULL (default)

Outputs the complete message description, including message text, facility name, explanation, user action, and user-supplied comment, if any.

/INSERT=*filename*.MSGHLP
/INSERT=TT:

Updates the first of the following files to be found with new or changed information from the specified .MSGHLP file, or, if TT: is specified, with the data entered immediately at the terminal:

- A .MSGHLP$DATA file specified with the **/LIBRARY** qualifier
- The first .MSGHLP$DATA file in a search path specified by the **/LIBRARY** qualifier
- The first .MSGHLP$DATA file in the default search path (defined by logical logical name MSGHLP$LIBRARY)
- SYS$HELP:MSGHLP$LIBRARY.MSGHLP$DATA (the default .MSGHLP$DATA file)

You must have write access for the .MSGHLP$DATA files supplied by VSI to insert data into these files. User-supplied data is identified by change bars in Help Message output.

/LIBRARY=*disk*:[*directory*]*filename*.MSGHLP$DATA
/LIBRARY=*disk*:[*directory*]
/LIBRARY=*logical-name*

Defines the messages database for the current command to be a particular .MSGHLP$DATA file, all the .MSGHLP$DATA files in a specified directory, or all the files in a search path defined by a logical name.

For most operations, the default database is either SYS$HELP:MSGHLP$LIBRARY.MSGHLP$DATA or a search path of .MSGHLP$DATA files defined by the logical name MSGHLP$LIBRARY.

For **/DELETE** and **/INSERT** operations, the default database is either SYS$HELP:MSGHLP$LIBRARY.MSGHLP$DATA or the first file in a search path defined by the logical name MSGHLP$LIBRARY.

/OUTPUT=*filespec*

Writes output to the specified file. By default, Help Message writes output to SYS$OUTPUT, which is normally the terminal. Use of **/OUTPUT=*filespec*** is incompatible with **/PAGE**.

/PAGE (default for screen display)
/NOPAGE

Displays terminal output one screen at a time. The page length is automatically set to one line less than the value specified by **SET TERMINAL/PAGE**. Use of **/PAGE** is incompatible with **/OUTPUT=*filespec***.

/SECTION_FILE=*
/SECTION_FILE=*file-spec*

Identifies the specified message section file to the system so that Help Message can interpret the $STATUS values for the messages in that file. The default file specification is SYS$MESSAGE:.EXE. Specifying **/SECTION_FILE=*** automatically includes all message section files supplied by OpenVMS. For more information, see the *OpenVMS System Messages: Companion Guide for Help Message Users*.

Note

The results of using this qualifier are entirely independent from those created by the **SET MESSAGE** command. The Help Message utility and Message utility do not interact. You must separately code each utility to obtain the results you want.

/SORT
/NOSORT (default)

Sorts output in alphabetical order. If a sort fails, retry the operation using the **/WORK_FILES** qualifier.

/STATUS=*status-code*
/STATUS='*symbol*'
/STATUS='$STATUS' (default)

Outputs the message corresponding to the specified status code. You can specify the status code with a decimal or hexadecimal number or a symbol enclosed in apostrophes. You can omit leading zeros, but you must prefix any hexadecimal number with "%X".

If a **HELP/MESSAGE** command does not include a search string, Help Message by default outputs the message corresponding to the CLI symbol $STATUS; that is, Help Message displays information on how the last executed command completed.

You cannot specify a search string or **/FACILITY** with **/STATUS**. **/FACILITY** is also illegal if you omit the search string and default to **/STATUS='$STATUS'**.

/WORD_MATCH= INITIAL_SUBSTRING (default)
/WORD_MATCH= WHOLE_WORD

/WORD_MATCH=INITIAL_SUBSTRING matches all words that begin with a word specified in the search string. The search string can contain multiple words to be matched. Only messages that match every word in the search string (in any order) are output.

/WORD_MATCH=WHOLE_WORD matches whole words only and refines your search to the exact words specified. For example, an exact search on ACC screens out dozens of other messages containing words that begin with the letters ACC.

/WORK_FILES=*nn*
/WORK_FILES=0 (default if qualifier is omitted)
/WORK_FILES=2 (default if qualifier is entered with no value)

Specifies that work files are to be used if the **/SORT** qualifier is specified. You can specify a value from 0 to 10 for *nn*. This qualifier has no effect if **/SORT** is not specified.

Examples

1. ```
   $ SHOW DEVICE KUDOS
   %SYSTEM-W-NOSUCHDEV, no such device available
   $ HELP/MESSAGE
   ```

   The first command creates an error. The default **HELP/MESSAGE** command (with no qualifiers) displays a description of the SYSTEM facility message NOSUCHDEV.

2. ```
   $ HELP/MESSAGE ACCVIO
   $ HELP/MESSAGE/BRIEF ACCVIO
   $ HELP/MESSAGE/FACILITY=SYSTEM ACCVIO
   $ HELP/MESSAGE VIRTUAL ACCESS
   $ HELP/MESSAGE/STATUS=12
   $ HELP/MESSAGE/STATUS=%XC
   ```

   These commands demonstrate how you can use various qualifiers to access and display the ACCVIO message in different formats.

3. ```
   $ HELP/MESSAGE/BRIEF ACC
   $ HELP/MESSAGE/BRIEF/WORD_MATCH=WHOLE_WORD ACC
   ```

   In the first command, Help Message by default matches dozens of words beginning with the string ACC. The **/WORD_MATCH=WHOLE_WORD** qualifier dramatically refines the search to match the exact word only.

4. ```
   $ HELP/MESSAGE/FACILITY=(BACKUP,SHARED)/SORT/OUTPUT=MESSAGES.TXT
   ```

   This command selects all messages issued by the BACKUP facility and those messages documented as "Shared by several facilities", alphabetizes them, and outputs them to a printable file called MESSAGES.TXT.

   By selecting the messages you want and directing them to a file, you can create and print your own customized messages documentation.

5. ```
   $ HELP/MESSAGE/EXTRACT=BADMESSAGE.MSGHLP BADMESSAGE
   $ HELP/MESSAGE/DELETE=BADMESSAGE.MSGHLP-
   _$ /LIBRARY=SYS$LOGIN:MYMESSAGES.MSGHLP$DATA
   $ CONVERT SYS$LOGIN:MYMESSAGES.MSGHLP$DATA-
   _$ SYS$LOGIN:MYMESSAGES.MSGHLP$DATA
   $ PURGE SYS$LOGIN:MYMESSAGES.MSGHLP$DATA
   $ HELP/MESSAGE/INSERT=BADMESSAGE.MSGHLP
   ```

   The first command in this sequence extracts the hypothetical message BADMESSAGE from the default database and outputs it to file BADMESSAGE.MSGHLP.

   The second command uses the BADMESSAGE.MSGHLP file to delete the BADMESSAGE description from the MYMESSAGES.MSGHLP$DATA file specified by the **/LIBRARY** qualifier.

   The next two commands compress the MYMESSAGES.MSGHLP$DATA file to save disk space after the deletion.

   The last command uses the BADMESSAGE.MSGHLP file (possibly an edited version at a later time) to insert the BADMESSAGE message into the default .MSGHLP$DATA file.

6. ```
   $ HELP/MESSAGE/EXTRACT=NOSNO.MSGHLP NOSNO
   $ EDIT/EDT NOSNO.MSGHLP
   ```

```
1NOSNO, can't ski; no snow
2XCSKI, XCSKI Program
3Your attempt to ski failed because there is no snow.
4Wait until there is snow and attempt the operation again.
5If you don't want to wait, go to a location where there is
5snow and ski there.
5
5Or, try ice skating instead!
[EXIT]
$ HELP/MESSAGE/INSERT=NOSNO.MSGHLP
```

This command sequence shows how users with write access to .MSGHLP$DATA files supplied by VSI can add a comment to a message.

The first command extracts hypothetical message `NOSNO` to file `NOSNO.MSGHLP`. The second command edits the `.MSGHLP` file to add a comment at the end of the message. Each comment line, even blank lines, includes a `5` prefix. The next command updates the database by using `NOSNO.MSGHLP` to insert the updated message into the default .MSGHLP$DATA file.

IF

IF — Tests the value of an expression and, depending on the syntax specified, executes given commands.

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 (&).

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 *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-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

Note

VSI advises against assigning a symbolic name that is already a DCL command name. VSI especially discourages the assignment of symbols such as IF, THEN, ELSE, and GOTO, which can affect the interpretation of command procedures.

The **IF** command tests the value of an expression and if the expression is true, 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

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.

3.
```
$  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.

4.
```
$ if 1 .eq. 1
$ then
$   if 2 .eq. 2
$   then
$     write sys$output  "Hello!"
$   endif
$ endif
```

This example shows how to use a nested **IF** structure.

INITIALIZE

INITIALIZE — Formats a disk or magnetic tape volume, writes a label on the volume, and leaves the disk empty except for the system files containing the structure information. All former contents of the disk are lost.

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 ANSI characters; for a magnetic tape volume, you can specify a maximum of 6 alphanumeric characters. Letters are automatically changed to uppercase. VSI strongly recommends that a disk volume label should only consist of alphanumeric characters, dollar signs ($), underscores (_), and hyphens (-).

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

Note

Requires VOLPRO (volume protection) privilege for most **INITIALIZE** command operations.

The default format for disk volumes in the OpenVMS 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 must have VOLPRO privilege to initialize a volume, except in the following cases:

- 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

After the volume is initialized and mounted, the **SET SECURITY** command maybe used to modify the security profile. When you initialize a disk volume, the caching attribute of its root directory (000000.DIR;1) is set to write-through. This means that by default, all the files and directories that you create in the volume will inherit a caching attribute of write-through. To change the caching attribute, use the **SETFILE** command with the **/CACHING_ATTRIBUTE** qualifier.

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 a magnetic tape successfully by repeating the **INITIALIZE** command from an account that has 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 VOLPRO privilege, the **INITIALIZE** command initializes a disk without reading the ownership information. If you do not have 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 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 (ODS-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 VSI and supported on the OpenVMS 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 *OpenVMS 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 16380 blocks, or 1/50th the volume size, whichever is smaller.

For Files-11 On-Disk Structure Level 5 (ODS-5) disks, the default cluster size is 16. In this case the minimum value allowed by the following equation is applied:

*(disk size in number of blocks)/(65535 * 4096)*

Any fractional values must be rounded up to the nearest integer and, by default, are rounded up to the next multiple of 16.

For Files-11 On-Disk Structure Level 2 (ODS-2) disks, the default cluster size depends on the disk capacity; disks with less than 50,000 have a default of 1. Disks that are larger than 50,000 have a default of either 16 or the result of the following formula, whichever is greater:

*(disk size in number of blocks)/(255 * 4096)*

Any fractional values must be rounded up to the nearest integer and, by default, are rounded up to the next multiple of 16.

Note

For Version 7.2 and later, you can specify a cluster size for ODS-2 volumes smaller than allowed by the ODS-2 formula; however, if you try to mount this volume on a system running a version prior to 7.2, the mount fails with the following error:

```
%MOUNT-F-FILESTRUCT, unsupported file structure level
```

If you choose the default during the initialization of an ODS-2 disk, your disk can be mounted on prior versions of OpenVMS.

For ODS-1 disks, the cluster size must always be 1.

Note

If you specify **/LIMIT** and do not specify a value for **/CLUSTER_SIZE**, a value of **/CLUSTER_SIZE=16** is used.

/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:

Option	Description
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*

Allows you to specify the format density value for certain tapes and disks.

For magnetic tape volumes, specifies the density in bits per inch (bpi) at which the magnetic tape is to be written. 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.

The **/DENSITY** qualifier does not apply to any TF tape device.

Valid tape density values are:

Keyword	Meaning
DEFAULT	Default density
800	NRZI 800 bits per inch (BPI)
1600	PE 1600 BPI
6250	GRC 6250 BPI
3480	IBM 3480 HPC 39872 BPI
3490E	IBM 3480 compressed
833	DLT TK50: 833 BPI
TK50	DLT TK50: 833 BPI
TK70	DLT TK70: 1250 BPI
6250	RV80 6250 BPI EQUIVALENT
NOTE: Only the keywords above are understood by TMSCP/TUDRIVER code prior to OpenVMS Version 7.2. The remaining keywords in this table are supported only on Alpha systems.	
TK85	DLT Tx85: 10625 BPI - Cmpt III - Alpha/IA-64 servers only

Keyword	Meaning
TK86	DLT Tx86: 10626 BPI - Cmpt III - Alpha/IA-64 servers only
TK87	DLT Tx87: 62500 BPI - Cmpt III - Alpha/IA-64 servers only
TK88	DLT Tx88: (Quantum 4000) - Cmpt IV - Alpha/IA-64 servers only
TK89	DLT Tx89: (Quantum 7000) - Cmpt IV - Alpha/IA-64 servers only
QIC	All QIC drives are drive-settable only - Alpha/IA-64 servers only
8200	Exa-Byte 8200 - Alpha/IA-64 servers only
8500	Exa-Byte 8500 - Alpha/IA-64 servers only
DDS1	Digital Data Storage 1 - 2G - Alpha/IA-64 servers only
DDS2	Digital Data Storage 2 - 4G - Alpha/IA-64 servers only
DDS3	Digital Data Storage 3 - 8-10G - Alpha/IA-64 servers only
DDS4	Digital Data Storage 4 - Alpha/IA-64 servers only
AIT1	Sony Advanced Intelligent Tape 1 - Alpha/IA-64 servers only
AIT2	Sony Advanced Intelligent Tape 2 - Alpha/IA-64 servers only
AIT3	Sony Advanced Intelligent Tape 3 - Alpha/IA-64 servers only
AIT4	Sony Advanced Intelligent Tape 4 - Alpha/IA-64 servers only
DLT8000	DLT 8000 - Alpha/IA-64 servers only
8900	Exabyte 8900 - Alpha/IA-64 servers only
SDLT	SuperDLT1 - Alpha/IA-64 servers only
SDLT320	SuperDLT320 - Alpha/IA-64 servers only

Note that tape density keywords cannot be abbreviated.

To format a diskette on RXnn diskette drives, use the **INITIALIZE/DENSITY** command. Specify the density at which the diskette is to be formatted as follows:

Keyword	Meaning
single	RX01 - 8 inch
double	RX02 - 8 inch
dd	double density: 720K - 3 1/2 inch
hd	high density: 1.44MB - 3 1/2 inch
ed	extended density: 2.88MB - 3 1/2 inch

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.

Note

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*

The effect of this qualifier depends on the disk structure:

- For ODS-1, **/DIRECTORIES** allows space for the specified number of directory entries to be reserved in 000000.DIR (the MFD).
- For ODS-2 and ODS-5, **/DIRECTORIES** allows the initial size of the MFD to be set. The specified number is divided by 16, to produce the number of blocks to preallocate. This number is then rounded up to a whole number of clusters.

The number-of-entries value must be an integer between 16 and 16000. The default value is 16.

/ERASE[=*keyword*]
/NOERASE (default)

Specifies whether to perform a data security erase (DSE) and, on disk volumes only, whether to set the volume characteristic to ERASE_ON_DELETE.

The **/ERASE** qualifier applies to Files-11 On-Disk Structure Level 2 (ODS-2) and Level 5 (ODS-5) disks 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.

For tape volumes, **/ERASE** physically destroys deleted data by writing over it.

For disk volumes, when **/ERASE** is specified with no keywords, this command does the following:

- Performs a data security erase (DSE) by writing the system-specified erase pattern into every block on the volume before initializing it. The amount of time taken by the DSE operation depends on the volume size.
- Sets the volume characteristic to ERASE_ON_DELETE so that each file on the volume will be erased by a DSE when it is deleted.

For disk volumes, two optional keywords allow you to independently specify just one of the actions noted above.

- **/ERASE=INIT**

  Performs a data security erase (DSE) operation on the volume before initializing it, but does *not* set the volume characteristic to ERASE_ON_DELETE. This operation takes longer than specifying **/ERASE=DELETE** and is equivalent to performing **SET VOLUME/NOERASE_ON_DELETE**.

- **/ERASE=DELETE**

  Sets the ERASE_ON_DELETE volume characteristic, but does *not* perform a DSE operation on the disk.

If neither (or both) keywords are specified, both actions are performed. That is, **/ERASE** is equivalent to **/ERASE=(INIT,DELETE)**.

/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 and Level 5 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.

The OpenVMS operating system uses the default volume extension 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 (ODS-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 *VSI OpenVMS Guide to System Security* [https://docs.vmssoftware.com/vsi-openvms-user-s-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 an OpenVMS system, but is provided to control the process's use of the volume on RSX-11M systems. OpenVMS systems always use the default file protection. Use the **SET PROTECTION/DEFAULT** command to change the default file protection.

/GPT (default for IA-64 servers)
/NOGPT (default for Alpha)

Applies to Files-11 On-Disk Structure Level 2 (ODS-2) and Level 5 (ODS-5) disks *only*.

Note

If you specify **/GPT**, the disk might not mount on some systems running older versions of OpenVMS.

When **/GPT** is specified, the system file [000000]GPT.SYS is created. GPT.SYS contains partition/boot information needed by the IA-64 console software. GPT is an abbreviation for GUID Partition Table, where GUID stands for Global Unique Identifier.

The BACKUP utility recognizes GPT.SYS and maintains its contents in a save/restore operation.

If **/NOGPT** is specified, the pre-Version 8.2 VBN layout of [000000]INDEXF.SYS is used. The VBN layout is described in the Guide to OpenVMS File Applications *[https://docs.vmssoftware.com/guide-to-openvms-file-applications/]* [https://docs.vmssoftware.com/guide-to-openvms-file-applications/] and in *VMS File System Internals* by Kirby McCoy (ISBN 1-55558-056-4, 1990).

/GROUP

Used in conjunction with the **/NOSHARE** qualifier to create a group volume. The group volume allows access by system (S), owner (O), and group (G) accessors. The protection is (S:RWCD,O:RWCD,G:RWCD,W).

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. However, if **/LIMIT** is specified and no value is specified for **/HEADERS** or **/MAXIMUM_FILES**, the following defaults apply:

- **/MAXIMUM_FILES**: 16711679 files
- **/HEADERS**: 0.5 percent of the size of the current device MAXBLOCK (an F$GETDVI item code)

  For example, for a 33GB disk, the default number of preallocated header blocks would be approximately 355000.

/HEADERS 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.

Note

The default value for the **/HEADERS** qualifier is generally insufficient for ODS-2 and ODS-5 disks. To improve performance and avoid SYSTEM-F-HEADERFULL errors, VSI recommends that you set this value to be approximately the number of files that you anticipate having on your disk; however, grossly overestimating this value will result in wasted disk space.

The **/HEADERS** qualifier controls how much space is initially allocated to INDEXF.SYS for headers. Each file on a disk requires at least one file header and each header occupies one block within INDEXF.SYS. Files that have many Access Control Entries (ACE) or are very fragmented may use more than one header.

The default value of 16 leaves room for less than 10 files to be created before INDEXF.SYS must extend; therefore, try to estimate the total number of files that will be created on the disk and specify it here. This will improve disk access performance. Overestimating the value may lead to wasted disks pace. This value cannot be changed without reinitializing the volume.

INDEXF.SYS is limited as to how many times it may extend. When the map area in its header (where the retrieval pointers are stored) becomes full, file creation fails with the message "SYSTEM-W-HEADERFULL".

/HIGHWATER (default)
/NOHIGHWATER

Applies to Files-11 On-Disk Structure Level 2 (ODS-2) and Level 5 (ODS-5) disks *only*.

Sets the file high-water 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.

/HOMEBLOCKS=*option*

Applies to Files-11 On-Disk Structure Level 2 (ODS-2) and Level 5 (ODS-5) disks *only*.

Specifies where the volume's home block and spare copy of the home block are placed on disk. The value of *option* can be one of the following:

- GEOMETRY

Causes the home blocks to be placed at separate locations on disk, to protect against failure of a disk block. Placement depends on the reported geometry of the disk.

- FIXED (default)

  Causes the home blocks to be placed at separate fixed locations on the disk. Placement is independent of the reported geometry of the disk. This caters to disks that report different geometries according to which type of controller they are attached to.

- CONTIGUOUS

  Causes the home blocks to be placed contiguously at the start of the disk. When used with the **/INDEX=BEGINNING** qualifier, this setting allows container file systems to maximize the amount of contiguous space on the disk, for example, to hold one large file, such as a database.

/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*

/INTERCHANGE

Specifies that the magnetic tape will be used for interchange in a heterogeneous vendor environment. The **/INTERCHANGE** qualifier omits the ANSI VOL2 labels. Under OpenVMS, the ANSI VOL2 labels contain OpenVMS specific security attributes.

For more information on the **/INTERCHANGE** qualifier and on magnetic tape labeling and tape interchange, see the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/].

/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 OpenVMS ANSI volume label VOL1 on an ANSI magnetic tape. The character maybe any valid ANSI "a" character. This set of characters includes numeric characters, uppercase letters, and any one of the following non-alphanumeric characters:

  ```
  ! " % ' ( ) * + , - . / : ; < = > ?
  ```

By default, the OpenVMS operating system provides a routine that checks this field in the following manner:

- If the magnetic tape was created on a version of the OpenVMS 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 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.

/LIMIT[=*n*]

Applies to Files-11 On-Disk Structure Level 2 (ODS-2) and Level 5 (ODS-5) disks *only*.

Specifies that the volume should be initialized with volume expansion. *n* defines the maximum growth potential of the volume in blocks. If no value is specified, the maximum expansion potential is set up.

The maximum value depends on the value specified for **/CLUSTER_SIZE**:

/CLUSTER_SIZE >= 16	1 TB of expansion is set up.
/CLUSTER_SIZE < 16	Expansion limit is set to 65535*4096*Cluster_value because the maximum size of the bitmap is 65535 blocks.

For more information about volume expansion, see the relevant section in the *VSI OpenVMS Volume Shadowing Guide* [https://docs.vmssoftware.com/vsi-openvms-volume-shadowing-guide/#dynamic_vol_extention_sec] manual.

The minimum allowed value is the largest of the following values:

- The value supplied with **/LIMIT**
- The physical disk size
- The size resulting from a 256-block BITMAP.SYS file (that is, 256 * 4096 bits/block * Disk Cluster Value)

If a value less than the minimum is supplied, the value is increased to the minimum. This value is displayed (in blocks) as the "Expansion Size Limit" in the output from a **SHOW DEVICE/FULL** command.

Note

If you specify **/LIMIT** and do not explicitly set a value for the following parameters, the defaults for these parameters are set as follows:

- **/CLUSTER_SIZE**: 16
- **/MAXIMUM_FILES**: 16711679 files

- **/HEADERS**: 0.5 percent of the size of the current device MAXBLOCK (an F$GETDVI item code)

  For example, for a 33GB disk, the default number of preallocated header blocks would be approximately 355000.

/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:

*(volume size in blocks)/((cluster factor + 1) * 2)*

Note

If **/LIMIT** is specified and no value is set for **/MAXIMUM_FILES**, the default is 16711679 files.

The maximum size you can specify for any volume is as follows:

(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 any device that supports data compaction. 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:

Option	Description
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, OpenVMS provides a routine that checks this field in the following manner. If the magnetic tape was created on a version of OpenVMS that conforms to Version 3 of ANSI, this option must be used to override any character other than an ASCII space. If a protection

Option	Description
	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 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.

You can initialize a volume previously initialized with **/PROTECTION** if you have control access.

/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 *VSI OpenVMS Guide to System Security* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#UIC_FORMAT_DIR].

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).
- Specify the *access* parameter as read (R), write (W), create (C), 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 an OpenVMS specific volume label. The system applies only read (R) and write (W) access restrictions; create 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 relevant section in the *VSI OpenVMS Guide to System Security* [https://docs.vmssoftware.com/vsi-openvms-guide-to-system-security/#PROTECT_CODE_DETAILS]. 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.

/SHADOW=(*device_name_1*, *device_name_2*, *device_name_3*) *label* (Alpha/IA-64 servers only)

Initializes multiple members of a future shadow set. Initializing multiple members in this way eliminates the requirement of a full copy when you later create a shadow set.

When both the **/SHADOW** and **/ERASE** qualifiers are specified, the **INITIALIZE** command performs the following operations:

- Formats up to six devices with one command, so that any three can be subsequently mounted together as members of a new host-based shadow set.
- Writes a label on each volume.
- Deletes all information from the devices except for the system files and leaves each device with identical file structure information. All former contents of the disks are lost.

VSI strongly recommends that you use the **/ERASE** qualifier. When **/ERASE** is specified, a merge operation is substantially reduced. However, using **/ERASE** has two side effects that are important considerations for volume shadowing: the setting of the ERASE volume attribute and the time it takes to initialize a volume using **/ERASE**.

If **/ERASE** is specified with **/SHADOW**, the disks are erased sequentially, which effectively doubles or triples the time it takes for the command to complete. If the disks are large, consider performing multiple, simultaneous **INITIALIZE/ERASE** commands (without **/SHADOW**) to erase the disks. Once all of those commands have completed, then execute an **INITIALIZE/SHADOW** command (without **/ERASE**).

Once you have initialized your devices using **/ERASE** and **/SHADOW**, you can then mount up to three of these devices as members of a new host-based shadow set.

Note that the **INITIALIZE/SHADOW** command should not be used to initialize a disk to be added to an existing shadow set, as no benefit is gained.

For more information about volume shadowing, see the *VSI OpenVMS Volume Shadowing Guide* [https://docs.vmssoftware.com/vsi-openvms-volume-shadowing-guide/].

/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.

/SIZE=*n*

When **/SIZE=*n*** is specified for a magnetic disk, *n* specifies the size (in blocks) of the logical volume (the space available for the file system). This allows you to **INITIALIZE** a disk with a file system size that is less than the physical volume size, which can be useful if you plan to create a shadow set using this disk and a smaller physical disk. The value of *n* is displayed (in blocks) as "Logical Volume Size" in the output from a **SHOW DEVICE/FULL** command.

For DECram disks, **/SIZE** specifies the size (in blocks) of the disk (device type DT$_RAM_DISK) to be allocated from available memory. The size of the device is created at disk initialization time.

To deallocate space, specify **/SIZE=0**. All resources specifically allocated to the DECram disk are returned to the system.

Note that *n* cannot exceed 524,280 blocks on versions of DECram prior to Version 2.3. DECram Version 2.3 running on an Alpha system supports up to 67,108,864 blocks, equivalent to 32GB.

/STRUCTURE=*level*

Specifies whether the volume should be formatted in Files-11 On-Disk Structure Level 1, 2 (the default), or 5.

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.

Note that Alpha does not support ODS-1 disks, and specifying 1 on Alpha results in an error.

See the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/] for more information about ODS-5 disks.

/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 to12 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.

/VOLUME_CHARACTERISTICS =([[NO]HARDLINKS,] [[NO]ACCESS_DATES[=*delta-time*]], [NO]SPECIAL_FILES)

Applies to Files-11 On-Disk Structure Level 5 (ODS-5) disks *only*.

Enables or disables hard links and automatic updates of access dates on ODS-5 volumes.

The default value for *delta-time* is 1 second, chosen to comply with the "seconds since EPOCH" time interface required by POSIX

```
st_atime
```

A site can choose a larger delta time to reduce overhead if 1-second granularity is not required.

Note that the NOACCESS_DATES option affects only the node on which the command is issued. Other nodes are not affected by the change until the next time the volume is mounted.

See the Guide to OpenVMS File Applications *[https://docs.vmssoftware.com/guide-to-openvms-file-applications/]* [https://docs.vmssoftware.com/guide-to-openvms-file-applications/] for additional information.

The volume characteristic [SPECIAL_FILES] allows you to disable symlinks. This eliminates file access failure audits that may occur due to symlinks being enabled for all processes in the current implementation.

/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:[GOLDSTEIN]
```

   This 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.

5.
```
$ INITIALIZE/ERASE/SHADOW=($4$DKA1300, $4$DKA1301) NONVOLATILE
$MOUN/SYS DSA42 /SHAD=( $4$DKA1300 , $4$DKA1301 ) NONVOLATILE
%MOUNT-I-MOUNTED, NONVOLATILE MOUNTED ON _DSA42:
%MOUNT-I-SHDWMEMSUCC, _$4$DKA1300: (WILD3) IS NOW A VALID MEMBER OF THE SHADOW SET
%MOUNT-I-SHDWMEMSUCC, _$4$DKA1301: (WILD4) IS NOW A VALID MEMBER OF THE SHADOW SET
$SHO DEV DSA42:

DEVICE              DEVICE          ERROR    VOLUME         FREE  TRANS MNT
```

```
 NAME               STATUS           COUNT     LABEL        BLOCKS COUNT CNT
DSA42:              MOUNTED              0  NONVOLATILE    5799600     1   1
$4$DKA1300: (WILD3) SHADOWSETMEMBER      0  (MEMBER OF DSA42:)
$4$DKA1301: (WILD4) SHADOWSETMEMBER      0  (MEMBER OF DSA42:)
```

This example shows correct use of the **INITIALIZE/ERASE/SHADOW** command. Note that the command specifies multiple devices on the same line.

INITIALIZE/QUEUE

INITIALIZE/QUEUE — Creates or initializes queues. You use this command to create queues and to assign them names and options. The **/BATCH** qualifier is required to create a batch queue.

Format

INITIALIZE/QUEUE *queue-name*[:]

Parameter

***queue-name*[:]**

Specifies the name of an execution queue or a generic queue. The queue name maybe 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

Use the **INITIALIZE/QUEUE** command to create a queue or to change the options of an existing queue that is stopped.

Note

Requires OPER (operator) privilege to create queues and manage(M) access to modify queues.

Normally you create output and batch queues by entering the necessary **INITIALIZE/QUEUE** commands when you set up your system or OpenVMS Cluster. Later, you can use the **INITIALIZE/QUEUE** command to create additional queues as they are needed. When you create a queue with the **INITIALIZE/QUEUE** command, information about the queue is stored in the queue database.

To create and start the queue at the same time, you can use the **INITIALIZE/QUEUE/START** command. If you want to create 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.

You can use the **INITIALIZE/QUEUE**, **START/QUEUE**, and **SET QUEUE** commands to change queue options; as you change queue options, information about the queue in the queue database is updated.

You can use the **INITIALIZE** and **START** commands only on stopped queues. To change options on a running queue, use the **SET QUEUE** command. To change queue options that cannot be altered with the **SET QUEUE** command, use the following procedure:

1. Stop the queue with the **STOP/QUEUE/NEXT** command.

2. Restart the queue with the **START/QUEUE** or the **INITIALIZE/QUEUE/START** command, specifying the appropriate qualifiers for the options you desire.

   Any qualifiers that you do not specify remain as they were when the queue was previously initialized, started, or set.

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
/NAME_OF_MANAGER

The following qualifiers apply to all types of execution queues:

/AUTOSTART_ON
/BASE_PRIORITY
/[NO]CHARACTERISTICS
/[NO]ENABLE_GENERIC
/[NO]NO_INITIAL_FF
/ON
/WSDEFAULT
/WSEXTENT
/WSQUOTA

The following qualifiers apply only to batch execution queues:

/CPUDEFAULT
/CPUMAXIMUM
/[NO]DISABLE_SWAPPING
/JOB_LIMIT

The following qualifiers apply only to printer, terminal, or server execution queues:

/[NO]BLOCK_LIMIT
/[NO]DEFAULT
/FORM_MOUNTED
/[NO]LIBRARY
/[NO]PROCESSOR
/[NO]RECORD_BLOCKING
/[NO]SEPARATE

Types of Queues

There are several different types of queues. 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.

Generic Queues

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 **/GENERIC** qualifier designates a queue as a generic queue. You specify the execution queues to which a generic queue feeds jobs in one of two ways:

- You can explicitly name execution queues assigned to the generic queue by including a list of queues with the **/GENERIC** qualifier.
- You can specify the execution queues that may receive jobs from any generic queue that does not specify an explicit target list by specifying the **/ENABLE=GENERIC** qualifier when you create the execution queue.

Generic queues, unlike execution queues, are not automatically stopped when the system is shut down or the queue manager is stopped; therefore, generic queues do not normally need to be restarted each time the system reboots.

Logical Queues

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 the assignment.

Execution Queues

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 hard copy 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 symbiont specified with the **/PROCESSOR** qualifier. Server queue symbionts are written by the customer.

Either the **/AUTOSTART_ON** qualifier or the **/ON** qualifier designates a queue as an execution queue, and specifies where the queue is to run.

By using the **/ON** qualifier, you can specify one node (for batch queues) or node and device (for output queues) on which the queue can be started. A queue initialized with the **/ON** qualifier needs to be started by a command explicitly naming the queue.

You can specify one or more nodes (or nodes and devices) on which the queue can be started by using the **/AUTOSTART_ON** qualifier. A queue initialized with the **/AUTOSTART_ON** qualifier is automatically started by the queue manager when any of the queue's nodes have been enabled for autostart by that queue manager.

Autostart Queues

An execution queue (either batch or output) can be designated as an autostart queue. Because all of a queue manager's autostart queues on a node can be started with a single command, autostart queues eliminate the need for lengthy queue startup procedures.

In an OpenVMS Cluster, autostart queues can be set up to run on one of several nodes. If a queue is set up this way, and the node on which the queue is running leaves the cluster, the queue can fail over to another node and remain available to the cluster.

The **/AUTOSTART_ON** qualifier designates an execution queue as an autostart queue.

Qualifiers

/AUTOSTART_ON=(*node*::[*device*][,...])

Designates the queue as an autostart execution queue and specifies the node, or node and device, on which the queue can be located. For batch queues, only *node* is applicable.

In a cluster, you can specify more than one node (or node and device) on which a queue can run, in the preferred order in which nodes should claim the queue. This allows the queue to fail over to another node if the node on which the queue is running leaves the cluster.

When you enter the **INITIALIZE/QUEUE** command with the **/AUTOSTART_ON** qualifier, you must initially activate the queue for autostart, either by specifying the **/START** qualifier with the **INITIALIZE/QUEUE** command or by entering a **START/QUEUE** command. However, the queue will not begin processing jobs until the **ENABLE AUTOSTART/QUEUES** command is entered for a node on which the queue can run.

This qualifier cannot be used in conjunction with the **/ON** or **/GENERIC** qualifier. However, if you are reinitializing an existing queue, you can specify the **/AUTOSTART_ON** qualifier for a queue previously created or started with the **/ON** qualifier. Doing so overrides the **/ON** qualifier and makes the queue an autostart queue.

For more information about autostart queues, see the relevant sections in the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/#CREATE_QUEUES].

/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 a null string ("") 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 system parameter PQL_MCPULM. The time cannot exceed the CPU time limit set by the **/CPUMAXIMUM** qualifier.

For information on specifying delta time, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT] or the online help topic Date. For more information on specifying CPU time limits, see *Table 1, "CPU Time Limit Specifications and Actions"*.

/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 system parameter PQL_MCPULM.

For information on specifying delta time, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT] or the online help topic Date. For more information on specifying CPU time limits, see *Table 1, "CPU Time Limit Specifications and Actions"*.

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 1, "CPU Time*

Limit Specifications and Actions" shows the action taken for each value specified and possible combinations of specifications.

Table 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:

Option	Description
[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.

Option	Description
[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. If no form type is explicitly specified with the FORM keyword, the system assigns the form DEFAULT to the queue. 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 options, see the description of the **/SEPARATE** qualifier. For more information on specifying default queue options, see the *VSI OpenVMS System Manager's Manual* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/].

/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 non-alphanumeric 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:

Option	Description
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 no form type is explicitly specified, the system assigns the form DEFAULT to the 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 and remains pending until the stock of the mounted form of the queue is identical to the stock of the form associated with the job.

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 and remains 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.

You cannot use the **/SEPARATE** qualifier with the **/GENERIC** qualifier.

/JOB_LIMIT=*n*

Indicates the number of batch jobs that can be executed concurrently from the queue. Specify a number in the range 1 to 65535. The job limit default value for *n* is 1.

/LIBRARY=*filename*
/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.

/NAME_OF_MANAGER=*name*

Identifies the name of the queue manager to control the queue. Once the queue is created, the queue manager assignment may not be altered.

If the **/NAME_OF_MANAGER** qualifier is omitted, then the default name SYS$QUEUE_MANAGER is used.

If the **INITIALIZE/QUEUE** command is used to modify a queue, and that queue is not controlled by the default queue manager, then the name of the controlling queue manager should be specified with the **/NAME_OF_MANAGER** qualifier. Alternately, the logical name SYS$QUEUE_MANAGER can be defined to be the correct queue manager, making that queue manager the default for the current process.

/NO_INITIAL_FF
/NONO_INITIAL_FF (default)

Allows user to specify whether a form feed should be sent to a printer device when a queue starts. To suppress the initial form feed, use the **/NO_INITIAL_FF** qualifier.

The **/NONO_INITIAL_FF** qualifier sends a form feed to the output device to ensure the paper is at the top of a page before printing begins.

/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.

You can specify an IP address and port number, in quotation marks, for the device. For more information about specifying IP addresses, see the TCP/IP Services for OpenVMS documentation.

The node name is used in OpenVMS Cluster systems; it must match the node name specified by the system parameter SCSNODE for the OpenVMS computer on which the queue executes.

You cannot use the **/ON** qualifier with the **/AUTOSTART_ON** or **/GENERIC** qualifier; however, if you are reinitializing an existing queue, you can specify the **/ON** qualifier for a queue previously created or started with the **/AUTOSTART_ON** qualifier. Doing so overrides the **/AUTOSTART_ON** option and makes the queue a nonautostart queue.

/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 *VSI OpenVMS Guide to System Security* [https://docs.vmssoftware.com/vsi-openvms-guide-to-system-security/#PROTECT_CODE_DETAILS]. The default UIC is [1,4].

/PROCESSOR=*filename*
/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).
- Specify the *access* parameter as read (R), submit (S), manage (M), or delete (D).

A null access specification means no access. The default protection is (SYSTEM:M, OWNER:D, GROUP:R, WORLD:S). If you include only one protection code, you can omit the parentheses. For more information on specifying protection codes, see the relevant section in the *VSI OpenVMS Guide to System Security* [https://docs.vmssoftware.com/vsi-openvms-guide-to-system-security/#PROTECT_CODE_DETAILS]. For more information on controlling queue operations through UIC-based protection, see the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/].

/RAD=*n*

Specifies the RAD number on which to run batch jobs assigned to the queue. The RAD value is validated as a positive integer between 0 and the value returned by the $GETSYI item code, SYI$_RAD_MAX_RADS.

RAD is supported on AlphaServer GS series systems and starting from OpenVMS Version 8.4, support is extended to NUMA capable IA-64 servers.

/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 OpenVMS 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 state after they have executed. The **/NORETAIN** qualifier enables you to reset the queue to the default. Possible options are as follows:

Option	Description
ALL (default)	Holds all jobs in the queue after execution.
ERROR	Holds in the queue only jobs that complete unsuccessfully.

A user can request a job retention option for a job by specifying the **/RETAIN** qualifier with the **PRINT**, **SUBMIT**, or **SET ENTRY** command; however, the job retention option you specify for a queue overrides any job retention option requested by a user for a job in that queue.

/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 not scheduled according to 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 options, 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:

Option	Description
[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 at the end of each job. 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 options that can be overridden, see the description of the **/DEFAULT** qualifier.

For more information on specifying mandatory queue options, see *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/].

/START
/NOSTART (default)

Starts the queue being initialized by the current **INITIALIZE/QUEUE** command.

For autostart queues, this qualifier activates the queue for autostart. The queue begins processing jobs when autostart is enabled with the **ENABLEAUTOSTART/QUEUES** command on any node on which the queue can run.

/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.

Specify the value of *n* as a number of 512-byte pagelets on Alpha systems. Note that OpenVMS rounds this value up to the nearest CPU-specific page so that the actual amount of physical memory allowed may be larger than the specified amount on Alpha. For further information, see the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/].

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 it includes a WSDEFAULT value).

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 2, "Working Set Default, Extent, and Quota Decision"*.

/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.

Specify the value of *n* as a number of 512-byte pagelets on Alpha. Note that OpenVMS rounds this value up to the nearest CPU-specific page so that the actual amount of physical memory allowed may be larger than the specified amount on Alpha.

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 it includes a WSEXTENT value).

You also can specify this qualifier for an output execution queue. Used in his 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 2, "Working Set Default, Extent, and Quota Decision"*.

/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.

Specify the value of *n* as a number of 512-byte pagelets on OpenVMS Alpha. OpenVMS rounds this value up to the nearest CPU-specific page so that the actual amount of physical memory allowed may be larger than the specified amount on OpenVMS Alpha. For further information, see the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/].

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 it includes a WSQUOTA value).

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 2, "Working Set Default, Extent, and Quota Decision"*) shows the action taken for different combinations of specifications that involve working set values.

Table 2. Working Set Default, Extent, and Quota Decision

Is the SUBMIT command value specified?	Is the queue value specified?	Action taken
No	No	Use the UAF value.
No	Yes	Use value for the queue.

Is the SUBMIT command value specified?	Is the queue value specified?	Action taken
Yes	Yes	Use smaller of the two values.
Yes	No	Compare specified value with UAF value; use the smaller.

Examples

1.
```
$ INITIALIZE/QUEUE/PROCESSOR=TELNETSYM -
_$ /ON="192.168.1.101:9100" SYS$PRINT
```

   This example initializes the SYS$PRINT print queue, specifying the TELNETSYM print symbiont, for the printer with the IP address 192.168.1.101 at TCP port 9100. For more information about TELNETSYM, see the relevant section in the *VSI TCP/IP Services for OpenVMS Management* [https://docs.vmssoftware.com/vsi-tcpip-services-for-openvms-6-management/#TELNET_PRINT_CHAP] manual.

2.
```
$ INITIALIZE/QUEUE/BATCH/START -
_$ /AUTOSTART_ON=(DATA::, WARF::, DEANNA::) BATCH_1
```

   The **INITIALIZE/QUEUE** command in this example creates the batch queue BATCH_1, and designates it as an autostart queue capable of executing on node DATA, WARF, or DEANNA. The **/START** qualifier activates the queue for autostart. The queue will begin executing on the first node (in the list of nodes specified) for which the **ENABLE AUTOSTART/QUEUES** command is entered.

   If the node on which BATCH_1 is executing is taken out of the OpenVMS Cluster, the queue will be stopped on that node and will fail over to the first available node in the node list on which autostart is enabled for a queue manager SYS$QUEUE_MANAGER.

   As long as autostart is enabled on one of the nodes in the list, this queue will be started and available to execute batch jobs. If all three nodes in the example are shut down or if autostart is disabled, the queue will remain stopped until one of the three nodes in the node list joins the cluster and executes the **ENABLE AUTOSTART/QUEUES** command.

   The **ENABLE AUTOSTART/QUEUES** and **INITIALIZE/QUEUE** commands affect only the queues managed by the default queue manager SYS$QUEUE_MANAGER because the **/NAME_OF_MANAGER** qualifier is not specified.

3.
```
$ 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 125 pages on OpenVMS Alpha (on a system with 8KB pages).

4.
```
$  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; therefore, 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.

5.
```
$ INITIALIZE/QUEUE/ON=QUEBID::/BATCH/RAD=0   BATCHQ1
$ SHOW QUEUE/FULL BATCHQ1
Batch queue BATCHQ1, stopped, QUEBID::
 /BASE_PRIORITY=4 /JOB_LIMIT=1 /OWNER=[SYSTEM]
 /PROTECTION=(S:M,O:D,G:R,W:S) /RAD=0
```

This example creates or reinitializes the batch queue BATCHQ1 to run on node QUEBID. All jobs assigned to this queue will run on RAD 0.

INQUIRE

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 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 single quotation marks (' ') 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

1.
```
$ 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

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 more information about the Install utility, see the relevant section in the *VSI OpenVMS System Management Utilities Reference Manual, Volume 1: A-L* [https://docs.vmssoftware.com/vsi-openvms-system-management-utilities-reference-manual-volume-i-a-l/#INSTALL_CH] or online help.

Format

INSTALL[*subcommand*] [*filespec*]

JAVA

JAVA — The **JAVA** command launches a Java™ application. It executes Java class files created by a Java compiler such as JAVAC.

Description

The **JAVA** command is available only if the Java Software Development Kit (SDK) or Run-Time Environment (RTE) is installed on your OpenVMS system.

Once the Java SDK or RTE is installed, you can access online help by entering this command:

```
$ JAVA -help
```

If the SDK documentation is installed on your OpenVMS system, you can use your browser to view documentation for the SDK tools (commands) and other reference material. For example, for the Java SDK v 1.4.0, point your browser to the following location:

```
SYS$COMMON:[JAVA$140.DOCS]INDEX.HTML
```

JOB

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.

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

Note

JOB cannot be abbreviated.

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 over punch), 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 absolute time or a combination of absolute and delta times. For complete information on specifying time values, see the relevant section in the *VSI OpenVMS User's*

Manual [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT] or the online help topic Date.

/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=*filename*

Specifies a different command language interpreter (CLI) with which to process the job. The *filename* 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 relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT] or the online help topic Date.

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 system 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 system 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 non blank 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).

Specify the value of *n* as a number of 512-byte pagelets on Alpha. Note that OpenVMS rounds this value up to the nearest CPU-specific page so that the actual amount of physical memory allowed may be larger than the specified amount on Alpha. The value *n* can be any integer from 1 to 65,535, 0, or the keyword NONE. For further information, see the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/].

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.

Specify the value of *n* as a number of 512-byte pagelets on Alpha. Note that OpenVMS rounds this value up to the nearest CPU-specific page so that the actual amount of physical memory allowed may be larger than the specified amount on Alpha. The value *n* can be any integer from 1 to 65,535, 0, or the keyword NONE. For further information, see the *VSI OpenVMS System Manager's Manual,*

Volume 1: Essentials [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/].

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*

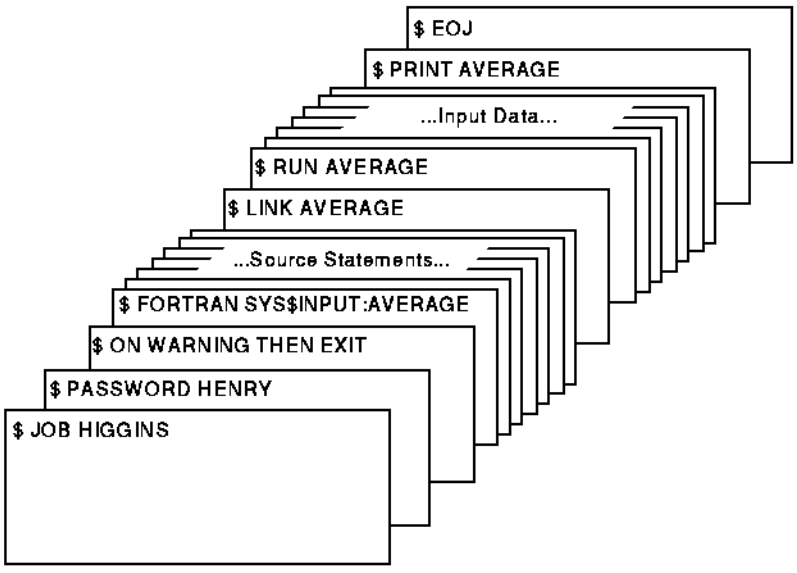
Defines the maximum working set size (working set quota) for the batch job; the **/WSQUOTA** qualifier overrides the value in the UAF.

Specify the value of *n* as a number of 512-byte pagelets on Alpha. Note that OpenVMS rounds this value up to the nearest CPU-specific page so that the actual amount of physical memory allowed may be larger than the specified amount on Alpha. The value *n* can be any integer from 1 to 65,535, 0, or the keyword NONE. For further information, see the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/].

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

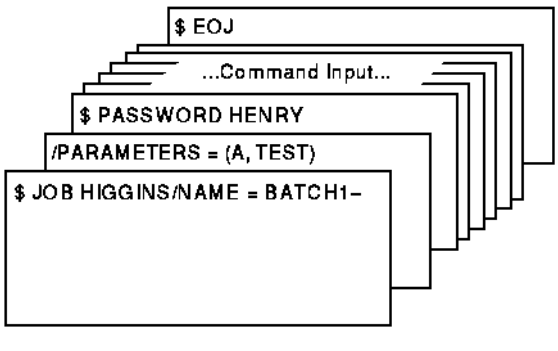
1.



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.

LIBRARY

LIBRARY — Invokes the Librarian utility, which creates, modifies, or describes an object, macro, help, text, or shareable image library. For more information about the Librarian utility, see the relevant section in the *VSI OpenVMS Command Definition, Librarian, and Message Utilities Manual* [https://docs.vmssoftware.com/vsi-openvms-command-definition-librarian-and-message-utilities/#LIB_CHAP] or online help.

Format

LIBRARY*library-filespec* [*input-filespec*[,...]]

LICENSE

LICENSE — Invokes the License Management utility, which manages software licenses on the OpenVMS operating system. For more information about the License Management utility, see the *VSI OpenVMS License Management Utility Guide* [https://docs.vmssoftware.com/vsi-openvms-license-management-utility-guide/] or online help.

Format

LICENSE *subcommand parameter*

LINK

LINK — Invokes the OpenVMS Linker, which links one or more object modules into a program image and defines execution characteristics of the image. For more information about the linker, including more information about the **LINK** command, see the *VSI OpenVMS Linker Utility Manual* [https://docs.vmssoftware.com/vsi-openvms-linker-utility-manual/] or online help.

Format

LINK ***filespec*****[,...]**

LOGIN Procedure

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 **Return**, **Ctrl/C**, or **Ctrl/Y** 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 **Return** 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 **Return**. The system displays the user name prompt again. Now retype your user name and press **Return** 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 login. 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.

/CONNECT (default)
/NOCONNECT

Specifies whether or not to reconnect to a virtual terminal.

/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.

/LOCAL_PASSWORD

Requests OpenVMS authentication using the user name and password information that is stored in the SYSUAF.DAT file. This qualifier is used to override external authentication if external authentication is unavailable.

/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, 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: HOFFMAN
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.

2.
```
Return
Username: HIGGINS/DISK=USER$
Password: <PASSWORD>
Welcome to OpenVMS Alpha (TM) Operating System, Version 7.3 on node LSR
    Last interactive login on Tuesday, 18-DEC-2001 08:41
    Last non-interactive login on Monday, 19-DEC-2001 15:43
$ SHOW DEFAULT
USER$:[HIGGINS]
```

   In this Alpha example, the **/DISK** qualifier requests that the default disk for the terminal session be USER$. The **SHOW DEFAULT** command shows that USER$ is the default disk.

3.
```
Return
Username:  JONES
Password: <PASSWORD>
User authorization failure
Return
Username: JONES
Password: <PASSWORD>
Welcome to OpenVMS Alpha (TM) Operating System, Version 7.3 on node LSR
        Last interactive login on Tuesday, 15-DEC-2001 09:16:47.08
        Last non-interactive login on Monday, 14-DEC-2001 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.

4.
```
Return
Username: JOYCE
Password: <PASSWORD>
Welcome to OpenVMS Alpha (TM) Operating System, Version 7.3 on node LSR
       Last interactive login on Tuesday, 15-DEC-2001 09:16:47.08
       Last non-interactive login on Monday, 14-DEC-2001 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 the *VSI OpenVMS DCL Dictionary: N–Z* [https://docs.vmssoftware.com/vsi-openvms-dcl-dictionary-n-z/#BRASS_75].

LOGOUT

LOGOUT — Terminates an interactive terminal session.

Format

LOGOUT

Description

You must use the **LOGOUT** command to end a terminal session. Undermost 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 dial up 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
  GILLINGS    logged out at 05-JUN-2001 17:48:56.73
```

   In this example, the **LOGOUT** command uses the default brief message form. No accounting information is displayed.

2.
```
$ LOGOUT/FULL
  GUZMAN      logged out at 05-JUN-2001 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

MACRO — By default on OpenVMS Alpha and OpenVMS Integrity servers, invokes the MACRO compiler for OpenVMS Systems to compile VAX assembly language source files into native OpenVMS Alpha or OpenVMS Integrity servers object code.

Format

MACRO *filespec*[,...]

Description

The **/ALPHA** qualifier causes the **MACRO** command to invoke the MACRO-64 assembler if it is installed on Alpha.

The **/MIGRATION** qualifier is the default on Alpha and Integrity servers On those platforms, specifying **MACRO** is the same as specifying **MACRO/MIGRATION**.

For a complete description of the MACRO compiler for OpenVMS Systems, see the *VSI OpenVMS MACRO Compiler Porting and User's Guide* [https://docs.vmssoftware.com/vsi-openvms-macro-compiler-porting-and-user-s-guide/].

MAIL

MAIL — Invokes the Mail utility, which is used to send messages to other users of the system. For more information about the Mail utility, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#MAIL_CH] or online help.

Format

MAIL [*filespec*] [*recipient-name*]

MERGE

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 more information about the Sort/Merge utility, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#SORT_CH] or online help.

Format

MERGE*input-filespec1*,*input-filespec2*[,...] *output-filespec*

MESSAGE

MESSAGE — Invokes the Message utility, which compiles one or more files of message definitions. For more information about the Message utility, see the relevant section in the *VSI OpenVMS Command Definition, Librarian, and Message Utilities Manual* [https://docs.vmssoftware.com/vsi-openvms-command-definition-librarian-and-message-utilities/#MSG_CHAP] or online help.

Format

MESSAGE *filespec*[,...]

MONITOR

MONITOR — Invokes the Monitor utility, which monitors classes of systemwide performance data at a specified interval. For more information about the Monitor utility, see the relevant section in the *VSI OpenVMS System Management Utilities Reference Manual, Volume 2: M-Z* [https://docs.vmssoftware.com/vsi-openvms-system-management-utilities-reference-manual-volume-ii-m-z/#d0e467] or online help.

Format

MONITOR[*/qualifier*[,...]] *classname*[,...] [*/qualifier*[,...]]

MOUNT

MOUNT — The Mount command (**MOUNT**) is used to make a disk or magnetic tape available for processing.

Format

MOUNT*device-name*[:][,...] [*volume-label*[,...]] [*logical-name*[:]]

Parameters

***device-name*[:][,...]**

Specifies the physical device name or logical name of the device on which the volume is to be mounted. On a system where volumes are not connected to HSCs (hierarchical storage controllers), use the following format:

ddcu:

The *dd* describes the device type of the physical devices used. For example, an RA60 disk drive is device type DJ, and an RA80 or RA81 disk drive is device type DU. The *c* identifies the controller, and the *u* identifies the unit number of the device.

On a system with HSCs, use one of the following formats:

node$ddcu:
allocation-class$ddcu:

If your devices are dual ported to HSCs, use the allocation-class format. For example, 125DUA23 represents an RA80 or RA81 disk with unit number 23. The disk's allocation class is 125. The *c* part of the format is always A for HSC disks. TROLL$DJA12 represents an RA60 disk with unit number 12. The device is connected to an HSC named TROLL. See the relevant section in the *VSI OpenVMS Cluster Systems Manual* [https://docs.vmssoftware.com/vsi-openvms-cluster-systems/#OVER_NAMING_H] for more information about naming conventions.

Device names can be generic so that if no controller or unit number is specified, the system attempts to mount the first available device that satisfies those specified components of the device names. If no volume is physically mounted on the specified device, **MOUNT** displays a message requesting that you place the volume in the device; after you place the volume in the named drive, **MOUNT** then completes the operation.

If you specify more than one device name for a disk or magnetic tape volume set, separate the device names with either commas or plus signs. For a magnetic tape volume set, you can specify more volume labels than device names or more device names than volumes.

volume-label[,...]

Specifies the label on the volume.

The number of characters allowed in a label depends on the type of device, as follows:

Device Type	Number of Characters in Label
Magnetic tape	0-6
Files-11 disk	1-12
ISO 9660 disk	1-32

OpenVMS requires disk volume labels to be unique in the first 12 characters within a given domain. For example, disks mounted by different members of the same group using the **/GROUP** qualifier must be unique. However, disks mounted in different domains, such as one mounted using the **/GROUP** qualifier and one mounted privately, can use the same volume label.

If you mount an ISO 9660 volume using the **/SYSTEM** or **/CLUSTER** qualifier, and the volume label is not unique within the first 12 characters, you must supply an alternate volume label using the qualifier **/OVERRIDE=IDENTIFICATION**. If you choose this option, then Mount verification is disabled for the device.

In addition, if a volume is part of a volume set and the first 12 characters of the volume-set name are the same as the first 12 characters of the volume label, a lock manager deadlock will occur. To avoid this problem, you must override either the volume label (by using the **/OVERRIDE** qualifier) or the volume-set name (by using the **/BIND** qualifier).

If you specify more than one volume label, separate the labels with either commas or plus signs. The volumes must be in the same volume set and the labels must be specified in ascending order according to relative volume number.

When you mount a magnetic tape volume set, the number of volume labels need not equal the number of device names specified. When a magnetic tape reaches the end-of-tape (EOT) mark, the system requests

the operator to mount the next volume on one of the devices. The user is not informed of this request; only the operator is informed.

When you mount a disk volume set, each volume label specified in the list must correspond to a device name in the same position in the device name list.

The volume-label parameter is not required when you mount a volume with the **/FOREIGN** or **/NOLABEL** qualifier or when you specify **/OVERRIDE=IDENTIFICATION**. To specify a logical name when you enter either of these qualifiers, type any alphanumeric characters in the volume-label parameter position.

logical-name[:]

Defines a 1- to 255-alphanumeric character string logical name to be associated with the volume.

If you do not specify a logical name, the **MOUNT** command assigns the default logical name DISK$*volume-label* to individual disk drives; it assigns the default logical name DISK$*volume-set-name* to the device on which the root volume of a disk volume set is mounted. Note that if you specify a logical name in the mount request that is different from DISK$*volume-label* or DISK$*volume-set-name*, then two logical names are associated with the device.

If you do not specify a logical name for a magnetic tape drive, the **MOUNT** command assigns only one logical name, TAPE$*volume-label*, to the first magnetic tape device in the list. No default logical volume-set name is assigned in this case.

The **MOUNT** command places the name in the process logical name table, unless you specify **/GROUP** or **/SYSTEM**. In the latter cases, it places the logical names in the group or system logical name table.

If you specify the **/CLUSTER** qualifier, the logical name is established on each node in the cluster.

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.

Do not use the logical name assigned to a volume as a distributed file system (DFS) access point. If you attempt to add a DFS access point using the same name as the logical name, DFS fails as in the following example:

```
$ SHOW LOG DISK$*
(LNM$SYSTEM_TABLE)
   "DISK$TIVOLI_SYS" = "TIVOLI$DUA0:"
$ RUN SYS$SYSTEM:DFS$CONTROL
DFS> ADD ACCESS DISK$TIVOLI_SYS TIVOLI$DUA0:[000000]
%DNS-W-NONSNAME, Unknown namespace name specified
```

If the logical name of a volume is in a process-private table, then the name is not deleted when the volume is dismounted.

Description

The Mount command (**MOUNT**) is used to make a disk or magnetic tape available for processing. **MOUNT** allows you to ensure that the device has not been allocated to another user, that a volume is physically loaded on the device specified, and that the label on the volume matches the label specified. For magnetic tape volumes, **MOUNT** also checks the volume accessibility field of the VOL1 label.

Normally, **MOUNT** allocates the device to the user who enters the command. However, mounting volumes with the **/SHARE**, **/GROUP**, or **/SYSTEM** qualifier deallocates the device, because the purpose of these qualifiers is to make the volume shareable.

Note

To mount a volume on a device, you must have read (R) or control (C) access to that device.

Any subprocess in the process tree can mount or dismount a volume for the job. When a subprocess mounts a volume (for the job) as private, the master process of the job becomes the owner of this device. This provision is necessary because the subprocess may be deleted and the volume should remain privately mounted for this job. However, when a subprocess explicitly allocates a device and then mounts a private volume on this device, the subprocess retains device ownership. In this situation, only subprocesses with SHARE privilege have access to the device.

Upon successful completion of the operation, **MOUNT** notifies you with a message sent to SYS$OUTPUT. If the operation fails for any reason, **MOUNT** notifies you with an error message.

Certain file utilities such as MOUNT allocate virtual memory to hold copies of the index file and storage bitmaps. Beginning with larger bitmaps in OpenVMS Version 7.2, the virtual memory requirements of these utilities increase correspondingly. To use MOUNT on volumes with large bitmaps, you might need to increase your page file quota. The virtual memory size is shown as Alpha 512-byte pagelets per block of bitmap. Note that the size of the index file bitmap in blocks is the maximum number of files divided by 4096. The virtual memory requirements for MOUNT is equal to the sum of the sizes of all index file bitmaps and storage bitmaps on the volume set. This requirement applies to MOUNT only if you rebuild a volume.

If you have a disk volume that you do not want the file system to cache, such as a database volume, use the **/NOCACHE** qualifier. This disables caching for the volume:

- It stops the following metadata caches from caching any metadata for the volume on the local node:

  The Extent Cache
  The File Identifier Cache
  The Quota Cache

- It stops the local Extended File Cache or Virtual I/O Cache from caching any files in the volume.

MOUNT Usage Summary

The Mount command (**MOUNT**) makes a disk or magnetic tape volume available for processing.

To invoke MOUNT, enter the DCL command **MOUNT**, followed by the device name, volume label, and logical name. You must include a device name and a volume label (unless you specify **/OVERRIDE=IDENTIFICATION** or use the **/FOREIGN** or **/NOLABEL** qualifier); the logical name is optional.

MOUNT returns you to the DCL level after it either successfully completes the operation or fails, generating an error message. If you press **Ctrl/Y** or **Ctrl/C**, MOUNT aborts the operation and returns you to the DCL prompt.

You can direct output from MOUNT operations with the **/COMMENT** and **/MESSAGE** qualifiers. When the mount operation requires operator assistance, use **/COMMENT** to specify additional information to

be included with the operator request. The **/COMMENT** text string is sent to the operator log file and to SYS$OUTPUT. The string must contain no more than 78 characters.

Use the **/MESSAGE** qualifier (this is the default) to send mount request messages to your current SYS $OUTPUT device. If you specify **/NOMESSAGE** during an operator-assisted mount, messages are not sent to SYS$OUTPUT; the operator sees them, however, if an operator terminal is enabled to receive messages.

Many **MOUNT** qualifiers require special privileges. Some qualifiers require different privileges according to which qualifier keyword you specify. See the individual qualifiers for details. The following table lists **MOUNT** qualifiers that require special privileges:

Qualifier	Keywords	Required Privilege
/ACCESSED		OPER
/CACHE=	[NO]DATA[=*n*]	OPER
	[NO]EXTENT[=*n*]	OPER
	[NO]FILE_ID[=*n*]	OPER
	[NO]QUOTA[=*n*]	OPER
/FOREIGN		VOLPRO[1]
/GROUP		GRPNAM
/MULTI_VOLUME		VOLPRO
/OVERRIDE=	ACCESSIBILITY	VOLPRO[1]
	EXPIRATION	VOLPRO[1]
	LOCK	VOLPRO[1]
	SHADOW	VOLPRO[1]
/OWNER_UIC=	*uic*	VOLPRO[1]
/PROCESSOR=	UNIQUE	OPER
	SAME:*device*	OPER
	file-spec	OPER and CMKRNL
/PROTECTION=	*code*	VOLPRO[1]
/QUOTA		VOLPRO[1]
/SYSTEM		SYSNAM
/WINDOWS=	*n*	OPER

[1]Or your UIC must match the volume UIC.

Qualifiers

/ACCESSED=*n*

Specifies, for ODS-1 disk volumes, the approximate number of directories that will be in use concurrently on the volume. The **/ACCESSED** qualifier is meaningless for ODS-2 volumes.

Specify a value from 0 to 255 to override the default that was specified when the volume was initialized.

You need the user privilege OPER to use **/ACCESSED**.

Example

The following command requests the volume labeled WORK to be mounted on DKA1, specifying 150 as the number of active directories on the volume:

```
$ MOUNT/ACCESSED=150 DKA1 WORK
```

/ASSIST (default)
/NOASSIST

Directs the mount operation to allow operator or user intervention if the mount request fails.

When you specify the **/ASSIST** qualifier, MOUNT notifies the user and certain classes of operator if a failure occurs during the mount operation. If a failure occurs, the operator or user can either abort the operation or correct the error condition to allow the operation to continue.

The operator-assist messages are sent to all operator terminals that are enabled to receive messages; magnetic tape mount requests go to TAPE and DEVICE operators, and disk mount requests go to DISK and DEVICE operators. Thus, if you need operator assistance while mounting a disk device, a message is sent to DISK operators. See the description of the **REPLY** command in the *VSI OpenVMS DCL Dictionary: N–Z* [https://docs.vmssoftware.com/vsi-openvms-dcl-dictionary-n-z/#BRASS_15] for more information about enabling and disabling operator terminals.

Any operator reply to a mount request is written to SYS$OUTPUT to be displayed on the user's terminal or written in a batch job log.

If no operator terminal is enabled to receive and respond to a mount assist request, a message is displayed informing the user of the situation. If a volume is placed in the requested drive, no additional operator response is necessary. If the mount request originates from a batch job and no operator terminal is enabled to receive messages, the mount is aborted. See the *OpenVMS System Messages: Companion Guide for Help Message Users* for a description of the error messages and their suggested user actions.

The default is **/ASSIST** and can be overridden by **/NOASSIST**.

Example

The following command mounts an HSG80 Fibre Channel disk volume labeled DOC and assigns the logical name WORK. The **/NOASSIST** qualifier signals **MOUNT** that no operator intervention is necessary.

```
$ MOUNT/NOASSIST $1$DGA0: DOC WORK
%MOUNT-I-MOUNTED, DOC          mounted on _$1$DGA0: (NODE)
```

/AUTOMATIC (default)
/NOAUTOMATIC

Determines whether MOUNT enables or disables automatic volume switching and labeling for magnetic tape or ISO 9660 CD-ROM.

Magnetic Tape

If you have multiple magnetic tape drives allocated to a volume set, the magnetic tape ancillary control process (MTACP) performs the volume switch by sequentially selecting the next available drive allocated to the volume set. The MTACP expects the next reel of the volume set to be loaded on that drive.

If the MTACP is writing to the volume set, it creates a label and initializes the magnetic tape with that label and the protections established for the first magnetic tape of the volume set. If it is reading from the volume set, the MTACP generates the label and attempts to mount the next magnetic tape with that label. If the drive has the wrong magnetic tape (or no magnetic tape) loaded, the MTACP sends a message to the operator's console to prompt for the correct magnetic tape.

The label generated by the MTACP fills the 6-character volume identifier field. The first four characters of the field contain the first four characters of the label specified in the **MOUNT** command, padded with underscores when the label is not at least four characters. The fifth and sixth characters contain the relative volume number for this reel in the volume set.

If you specify **/NOAUTOMATIC**, the MTACP requires operator intervention to switch to the next drive during end-of-tape processing, and requires that the operator specify a label for each new reel added to a volume set.

ISO 9660 CD-ROM

Under ISO 9660, not all volume-set members must be mounted to perform I/O operations against that volume set. By default, if I/O operations attempt to access an unmounted volume-set member, an operator message is sent to all DISK CLASS operators for system-mounted volume sets, or the owning process for privately mounted volume sets. The message specifies the volume-set member to mount to complete the I/O operation requested. If **/NOAUTOMATIC** is specified, then an I/O operation to a non mounted volume set member completes with an error message SS$_DEVNOTMOUNT.

Example

The following command instructs MOUNT not to generate its own label for the second volume, but to use the ones supplied with the **MOUNT** command instead. If the second volume is not already labeled, then the operator must use **REPLY/INIT** and supply the second label.

```
$ MOUNT/NOAUTOMATIC MTA0: ABCD,EFGH
```

/BIND=*volume-set-name*

Creates a volume set of one or more disk volumes or adds one or more volumes to an existing volume set.

The parameter, *volume-set-name*, specifies a 1- to 12-alphanumeric-character name identifying the volume set.

An ISO 9660 volume-set name can be from 1 to 128 characters in length.

OpenVMS requires volume-set names to be unique in the first 12 characters. In addition, if the first 12 characters of volume-set name are the same as the first 12 characters of any volume label, a lock manager deadlock will occur. To avoid this problem, you must override either the volume label (by using the **/OVERRIDE** qualifier) or the volume-set name (by using the **/BIND** qualifier).

You must specify the **/BIND** qualifier when you first create the volume set or each time you add a volume to the set. To dismount an individual volume of the volume set, you must use the **DISMOUNT** qualifier **/UNIT**; otherwise, dismounting an individual volume dismounts the entire volume set.

When you create a volume set, the volumes specified in the volume-label list are assigned relative volume numbers based on their positions in the label list. The first volume specified becomes the root volume of the set.

When you add a volume or volumes to a volume set, the first volume label specified must be that of the root volume, or the root volume must already be on line.

Note that if you attempt to create a volume set from two or more volumes that already contain files and data, the file system does not issue an error message when you issue the **MOUNT/BIND** command. However, the volumes are unusable as a volume set because the directory structures are not properly bound.

If you mount an ISO 9660 volume using the **/SYSTEM** or **/CLUSTER** qualifier, and the volume label is not unique within the first 12 characters, you must supply an alternate 12-character volume label using the qualifier ***/BIND=volume-set-name***. If you choose this option, then Mount verification is disabled for the device.

Note

Once a volume is bound into a volume set, it cannot easily be unbound. To unbind a bound volume set (BVS):

1. Do an image backup of the BVS.
2. Initialize all volumes of the BVS.
3. Do an image restore to a single volume with the **/NOINITIALIZE** qualifier, or do a non image restore to a single volume.

Examples

The following command creates a volume set named `LIBRARY`. This volume set consists of the volumes labeled `BOOK1`, `BOOK2`, and `BOOK3`, which are mounted physically on devices `DMA0`, `DMA1`, and `DMA2`, respectively.

```
$ MOUNT/BIND=LIBRARY  DMA0:,DMA1:,DMA2:  BOOK1,BOOK2,BOOK3
```

The following command creates a volume set with the logical name `TEST3`. The volume set `TEST3` is not shadowed, however each element of the volume set (`TEST3011` and `TEST3012`) is a shadow set, providing redundancy for the volume set as a whole.

```
$ MOUNT/BIND=TEST3 DSA3011/SHADOW=($1$DUA402:,$1$DUA403:),
DSA3012/SHADOW=($1$DUA404:,$1$DUA405:) TEST3011,TEST3012 TEST3
```

/BLOCKSIZE=*n*

Specifies the default block size for magnetic tape volumes.

The parameter, *n*, specifies the default block size value for magnetic tape volumes. Valid values are in the range 20 to 65,532 for OpenVMS RMS operations, and 18 to 65,534 for non OpenVMS RMS operations. By default, records are written to magnetic tape volumes in 2048-byte blocks. For foreign or unlabeled magnetic tapes, the default is 512 bytes.

You must specify **/BLOCKSIZE** in two situations:

- When mounting magnetic tapes that do not have HDR2 labels. For these magnetic tapes, you must specify the block size. For example, you must specify **/BLOCKSIZE=512** to mount an RT-11 magnetic tape.

- When mounting magnetic tapes that contain blocks whose sizes exceed the default block size (2048 bytes). In this case, specify the size of the largest block for the block size.

Example

In the following example, the **/BLOCKSIZE** qualifier specifies a block size of `1000` bytes; the default for a magnetic tape mounted with the **/FOREIGN** qualifier is 512.

```
$ MOUNT/FOREIGN/BLOCKSIZE=1000 MTA1:
```

/CACHE=(*keyword*[,...])
/NOCACHE

For disks, controls whether caching limits established at system generation time are disabled or overridden. With the TAPE_DATA option, enables write caching for the tape controller specified (if the tape controller supports write caching).

The following table lists the keywords for this qualifier:

Keyword	Description
DATA and NODATA	Enable or disable Extended File Caching (XFC). To enable XFC caching, you must specify the DATA (this is the default value for **/CACHE** qualifier).To disable XFC, specify NODATA. Note that **/NOCACHE** is equivalent to **/CACHE=NODATA**.
EXTENT[=*n*] and NOEXTENT	Enable or disable extent caching. To enable extent caching, you must have the operator user privilege (OPER) and you must specify *n*, the number of entries in the extent cache. Note that NOEXTENT is equivalent to EXTENT=0; both disable extent caching.
FILE_ID[=*n*] and NOFILE_ID	Enable or disable file identification caching. To enable file identification caching, you must have the operator user privilege (OPER) and you must specify *n*, the number of entries, as a value greater than 1. Note that NOFILE_ID is equivalent to FILE_ID=1; both disable file identification caching.
LIMIT=*n*	Specifies the maximum amount of free space in the extent cache in one-thousandths of the currently available free space on the disk.
QUOTA[=*n*] and NOQUOTA	Enable or disable quota caching. To enable quota caching, you must have the operator user privilege (OPER) and you must specify *n*, the number of entries in the quota cache. Normally *n* is set to the maximum number of active users expected for a disk with quotas enabled. Both NOQUOTA and QUOTA=0 disable quota file caching.
TAPE_DATA	Enables write caching for a magnetic tape device if the tape controller supports write caching. The **/CACHE** qualifier is the default for mounting tape devices. You must specify TAPE_DATA to enable write caching. If the tape controller does not support write caching, the keyword is ignored. The write buffer stays enabled even after you dismount the magnetic tape. To disable the write buffer, mount a tape with the **/NOCACHE** qualifier. If a tape supports compaction, then the default is compaction, and caching is enabled. For tape storage devices that support compaction, the following command is valid:

Keyword	Description
	`$ MOUNT TAPE_DATA/FOREIGN/MEDIA=NOCOMPACTION/ NOCACHE`
WRITETHROUGH	Disables the deferred write feature for file headers. By default, this feature is enabled, which improves the performance of applications, such as PATHWORKS, that use it. The deferred write feature is not available on Files-11 ODS-1 volumes.

Note

In a mixed-version OpenVMS cluster, an attempt to mount a volume with **/CLUSTER** and **/CACHE=[NO]DATA** from a V8.4 system fails on the pre-V8.4 systems (%MOUNT-W-RMTMNTFAIL) with MOUNT-F-BADPARAM.

Used with the disk options, the **/CACHE** qualifier overrides one or more of the present disk caching limits established at system generation time. Used with the TAPE_DATA option, the **/CACHE** qualifier enables write caching for the tape controller specified.

If you do not specify the **/CACHE** qualifier and it is not implied by the use of the qualifier **/MEDIA_FORMAT=COMPACTION**, caching is enabled by default.

If you specify more than one option, separate them by commas and enclose the list in parentheses. The options [NO]EXTENT, [NO]FILE_ID, LIMIT, and [NO]QUOTA apply only to a disk device. The option TAPE_DATA applies only to a tape device.

The **/NOCACHE** qualifier is effective only if compaction is not enabled. If compaction is enabled (with the **/MEDIA_FORMAT=COMPACTION**), caching is enabled by default.

If you specify **/NOCACHE** for a disk device, all caching is disabled for this volume. Note that the **/NOCACHE** qualifier is equivalent to **/CACHE=(NOEXTENT, NOFILE_ID, NOQUOTA, WRITETHROUGH, NODATA)**.

In the following command, NODATA is taken as default when you supply the following qualifiers `NOEXTENT`, `NOFILE_ID`, `NOQUOTA`, `WRITETHROUGH` (that is, XFC is disabled):

```
$ MOUNT/CACHE=(NOEXTENT, NOFILE_ID, NOQUOTA, WRITETHROUGH)
_$ $1$DGA0: FILES WORK
%MOUNT-I-MOUNTED, FILES mounted on $1$DGA0: (NODE)
```

In the following command, DATA is take as default (that is, XFC is enabled):

```
$ MOUNT/CACHE=(FILE_ID=10)
_$ $1$DGA0: FILES WORK
%MOUNT-I-MOUNTED, FILES mounted on $1$DGA0: (NODE)
```

If you specify **/NOCACHE** for a magnetic tape device, the tape controller's write cache is disabled for this volume.

Examples

The following command mounts an HSG80 Fibre Channel disk device labeled `FILES` and assigns the logical name `WORK`. The **/CACHE** qualifier enables an extent cache of `60` entries, a file identification cache of `60` entries, and a quota cache of `20`; it disables write back caching of file headers.

```
$ MOUNT/CACHE=(EXTENT=60,FILE_ID=60,QUOTA=20,WRITETHROUGH) -
_$ $1$DGA0: FILES WORK
%MOUNT-I-MOUNTED, FILES mounted on _$1$DGA0: (NODE)
```

The following command mounts the volume TAPE on device MUA0 and instructs MOUNT to enable the tape controller's write cache for MUA0:

```
$ MOUNT/CACHE=TAPE_DATA MUA0:  TAPE
%MOUNT-I-MOUNTED, TAPE  mounted on _NODE$MUA0:
```

The following command enables data cache (XFC) on a disk. The **/CACHE=DATA** qualifier is the default value for a basic **MOUNT** command:

```
$ MOUNT/CACHE=(DATA)
_$ $1$DGA0: FILES WORK
%MOUNT-I-MOUNTED, FILES mounted on $1$DGA0: (NODE)
```

The following command disables data cache (XFC) on a disk. **/NOCACHE** qualifier is equivalent to **/CACHE=(NODATA)**:

```
$ MOUNT/CACHE=(NODATA)
_$ $1$DGA0: FILES WORK
%MOUNT-I-MOUNTED, FILES mounted on $1$DGA0: (NODE)
```

The following command disables data cache that is, XFC and metadata cache that is, XQP. **/NOCACHE** qualifier is equivalent to **/CACHE=(NODATA)**:

```
$ MOUNT/NOCACHE
_$ $1$DGA0: FILES WORK
%MOUNT-I-MOUNTED, FILES mounted on $1$DGA0: (NODE)
```

/CLUSTER

Specifies that after the volume is successfully mounted on the local node, or if it is already mounted **/SYSTEM** on the local node, it is to be mounted on every other node in the existing OpenVMS Cluster (that is, the volume is mounted clusterwide).

Only system or group volumes can be mounted clusterwide. If you specify the **/CLUSTER** qualifier with neither the **/SYSTEM** nor the **/GROUP** qualifier, the default is **/SYSTEM**. Note that you must use a cluster device-naming convention. Use either *node$device-name* or *allocation-class$device-name* as required by your configuration.

You need the user privileges GRPNAM and SYSNAM, respectively, to mount group and system volumes clusterwide.

If the system is not a member of an OpenVMS Cluster, the **/CLUSTER** qualifier has no effect

Example

The following **MOUNT/CLUSTER** command mounts the volume SNOWWHITE on DOPEY$DMA1, then proceeds to mount the volume clusterwide. The **SHOWDEVICE/FULL** command displays information about the volume, including the other nodes on which it is mounted.

```
$ MOUNT/CLUSTER DOPEY$DMA1: SNOWWHITE DWARFDISK
%MOUNT-I-MOUNTED, SNOWWHITE         mounted on _DOPEY$DMA1:
$ SHOW DEVICE/FULL DWARFDISK:

Disk $2$DMA1: (DOPEY), device type RK07, is online, mounted,
```

```
   file-oriented device, shareable, served to cluster via MSCP
   Server, error logging is enabled.

   Error count                0  Operations completed               159
   Owner process             ""  Owner UIC                      [928,49]
   Owner process ID    00000000  Dev Prot         S:RWED,O:RWED,G:RW,W:R
   Reference count            1  Default buffer size                512
   Total blocks           53790  Sectors per track                   22
   Total cylinders          815  Tracks per cylinder                  3
   Allocation class           2

   Volume label     "SNOWWHITE"  Relative volume number               0
   Cluster size               3  Transaction count                    1
   Free blocks            51720  Maximum files allowed             6723
   Extend quantity            5  Mount count                          7
   Mount status          System  Cache name      "_$255$DWARF1:XQPCACHE"
   Extent cache size         64  Maximum blocks in extent cache    5172
   File ID cache size        64  Blocks currently in extent cache     0
   Quota cache size          25  Maximum buffers in FCP cache       349

  Volume status: ODS-2, subject to mount verification,
     file high-water marking, write-through XQP caching enabled,
     write-through XFC caching enabled.
  Volume is also mounted on DOC, HAPPY, GRUMPY, SLEEPY, SNEEZY, BASHFUL.
```

/COMMENT=*string*

Specifies additional information to be included with the operator request when the mount operation requires operator assistance.

The parameter, *string*, specifies a text string that is output to the operator log file and the current SYS$OUTPUT device. The string must contain no more than 78 characters.

Examples

The following command requests the operator to mount the disk volume TESTSYS on the device DYA1. Notice that the **/COMMENT** qualifier is used to inform the operator of the location of the volume. After the operator places the volume in DYA1, MOUNT retries the operation. After the operation completes, the operator request is canceled.

```
$ MOUNT DYA1:  TESTSYS/COMMENT="Volume in cabinet 6."
%MOUNT-I-OPRQST, Please mount volume TESTSYS in device _DYA1:
Volume in cabinet 6.
%MOUNT-I-MOUNTED TESTSYS    mounted on _DYA1:
%MOUNT-I-OPRQSTDON, operator request canceled - mount
completed successfully
```

The following command is the same as in the previous example. However, in this example, because the requested device is in use, the operator aborts the mount.

```
$ MOUNT DYA1:  TESTSYS/COMMENT="Volume in cabinet 6."
%MOUNT-I-OPRQST, Please mount volume TESTSYS in device _DYA1:
Volume in cabinet 6.
%MOUNT-I-OPREPLY, This is a '/pending' response from the operator.
31-DEC-1990 10:27:38.15, request 2 pending by operator TTB6
%MOUNT-I-OPREPLY, This is a '/abort' response from the operator.
31-DEC-1990 10:29:59.34, request 2 aborted by operator TTB6
%MOUNT-F-OPRABORT, mount aborted by operator
```

The following command requests the operator to mount the volume TESTSYS on the device DYA0. In this example, the operator notices that the requested device is in use and redirects the mount to device DYA1.

```
$ MOUNT DYA0:  TESTSYS/COMMENT="Volume in cabinet 6, once again with
 feeling."
%MOUNT-I-OPRQST, Please mount volume TESTSYS in device _DYA0:
Volume in cabinet 6, once again with feeling.
%MOUNT-I-OPREPLY, Substitute DYA1:
31-DEC-1990 10:43:42.30, request 3 completed by operator TTB6
%MOUNT-I-MOUNTED, TESTSYS    mounted on _DYA1:
```

/CONFIRM ***virtual-unit-name*****[:] /SHADOW=(*****physical-dev-name*****[:][,...])**
/NOCONFIRM ***virtual-unit-name*****[:] /SHADOW=(*****physical-dev-name*****[:][,...])**

Causes MOUNT to pause and request confirmation before performing a copy operation on the specified disk device. This qualifier is applicable only if you have the volume shadowing option. See the *VSI OpenVMS Volume Shadowing Guide* [https://docs.vmssoftware.com/vsi-openvms-volume-shadowing-guide/] for additional information.

This qualifier controls whether MOUNT issues a request to confirm a full copy operation when mounting a shadow set. The **/SHADOW** qualifier must be used with the **/CONFIRM** qualifier. Use **/CONFIRM** to display the volume label and volume owner for any specified physical device that is a target for a copy operation. MOUNT stops before any copy operations occur and issues the following prompt:

```
Allow FULL shadow copy on the above member(s)? [N]:
```

If you respond Y or YES, the mount operation continues automatically with copy operations allowed. If you respond N, NO, **Return**, or **Ctrl/Z**, the command quits without mounting any of the specified volumes (including volumes that did not require copy operations). If you type a response other than those listed above, MOUNT reissues the prompt.

The **/CONFIRM** qualifier is similar to **/NOCOPY**. Use **/CONFIRM** to mount shadow sets interactively; use **/NOCOPY** in the site-specific startup command procedure SYS$MANAGER:SYSTARTUP_VMS.COM.

Example

The following example shows how to use the **/CONFIRM** qualifier to check the status of potential shadow set members before any data is erased. The command instructs MOUNT to build a shadow set with the specified devices, and prompts for permission to perform a copy operation. The response of YES instructs MOUNT to mount the shadow set.

```
$ MOUNT/CONFIRM DSA0:/SHADOW=($200$DKA200:,$200$DKA300:,$200$DKA400:) X5OZCOPY

%MOUNT-F-SHDWCOPYREQ, shadow copy required
Virtual Unit - DSA0                      Volume Label - X5OZCOPY
     Member                    Volume Label Owner UIC
     $200$DKA200: (VIPER1)     X5OZCOPY     [SYSTEM]
     $200$DKA400: (VIPER1)     X5OZCOPY     [SYSTEM]
Allow FULL shadow copy on the above member(s)? [N]:)
Y

%MOUNT-I-MOUNTED, X5OZCOPY mounted on _DSA0:
%MOUNT-I-SHDWMEMSUCC, _$200$DKA300: (VIPER1) is now a valid member of
the shadow set
%MOUNT-I-SHDWMEMCOPY, _$200$DKA200: (VIPER1) added to the shadow set
with a copy operation
%MOUNT-I-SHDWMEMCOPY, _$200$DKA400: (VIPER1) added to the shadow set
with a copy operation
```

/COPY *virtual-unit-name*[**:**] **/SHADOW=(***physical-dev-name***[:][,...]) (default)**
/NOCOPY *virtual-unit-name*[**:**] **/SHADOW=(***physical-dev-name***[:][,...])**

Enables or disables copy operations on physical devices specified when you mount a shadow set. This qualifier is applicable only if you have the volume shadowing option. See the *VSI OpenVMS Volume Shadowing Guide* [https://docs.vmssoftware.com/vsi-openvms-volume-shadowing-guide/] for additional information.

The **/COPY** qualifier instructs MOUNT to perform copy operations on shadow set members. You can mount shadow sets with **/NOCOPY** to test if proposed shadow set members are targets of copy operations. If any of the specified volumes is a target of a copy operation, the command quits without mounting any of the specified volumes (including those that did not require a copy operation).

The **/NOCOPY** qualifier is similar to **/CONFIRM**. Use **/NOCOPY** to mount shadow sets in the site-specific startup command procedure SYS$MANAGER:SYSTARTUP_VMS.COM; use **/CONFIRM** for interactive mounting.

Example

The following example shows how to use the **/NOCOPY** qualifier to check the status of potential shadow set members before any data is erased. The command instructs MOUNT to build a shadow set with the specified devices only if a copy operation is not required. Because the device DUA7 required a copy operation to become a member of the shadow set, the mount failed. You could reissue the command specifying **/COPY** to instruct MOUNT to build the shadow set providing the necessary copy operation.

```
$ MOUNT/NOCOPY DSA2: /SHADOW=($1$DUA4:,$1$DUA6:,$1$DUA7:) -
_$  SHADOWVOL DISK$SHADOWVOL
%MOUNT-F-SHDWCOPYREQ, shadow copy required
%MOUNT-I-SHDWMEMFAIL, DUA7: failed as a member of the shadow set
%MOUNT-F-SHDWCOPYREQ, shadow copy required
```

/DATA_CHECK[=(*keyword***[,...])]**

Overrides the read-check or write-check option (or both) specified for a volume when it was initialized.

The keyword, READ, performs checks following all read operations, and the keyword, WRITE, performs checks following all write operations.

You can specify either or both of the keywords. If you specify more than one keyword, separate them by commas and enclose the list in parentheses.

If you specify the **/DATA_CHECK** qualifier without specifying a keyword, MOUNT defaults to **/DATA_CHECK=WRITE**.

Example

The following command mounts a volume labeled SAM on CLEMENS$DKA2 and assigns the logical name BOOK. The **/DATA_CHECK=READ** qualifier overrides a previous **INITIALIZE/DATA_CHECK=WRITE** specification, so that subsequent read operations on BOOK are subject to data-checking operations.

```
$ MOUNT/DATA_CHECK=READ CLEMENS$DKA2: SAM  BOOK
```

/DENSITY=*keyword*

Specifies the density at which a magnetic tape is to be written. This qualifier is valid only if you mount a tape specifying the **/FOREIGN** qualifier. If you change the density on a tape, the first operation on the tape must be a write operation.

The densities supported for tapes are shown in the following table:

Table 3. Keywords for Tapes

Keyword	Meaning
DEFAULT	Default density
800	NRZI 800 bits per inch (BPI)
1600	PE 1600 BPI
6250	GRC 6250 BPI
3480	IBM 3480 HPC 39872 BPI
3490E	IBM 3480 compressed
833	DLT TK50: 833 BPI
TK50	DLT TK50: 833 BPI
TK70	DLT TK70: 1250 BPI
6250	RV80 6250 BPI EQUIVALENT
NOTE: Only the symbols listed above are understood by TMSCP/TUDRIVER code prior to OpenVMS Version 7.2. The remaining symbols in this table are supported only on OpenVMS Alpha and Integrity server systems.	
TK85	DLT Tx85: 10625 BPI -- Cmpt III - Alpha/Integrity servers only
TK86	DLT Tx86: 10626 BPI -- Cmpt III - Alpha/Integrity servers only
TK87	DLT Tx87: 62500 BPI -- Cmpt III - Alpha/Integrity servers only
TK88	DLT Tx88: (Quantum 4000) -- Cmpt IV - Alpha/Integrity servers only
TK89	DLT Tx89: (Quantum 7000) -- Cmpt IV - Alpha/Integrity servers only
QIC	All QIC drives are drive-settable only - Alpha/Integrity servers only
TK85	DLT Tx85: 10625 BPI -- Cmpt III - Alpha/Integrity servers only
TK86	DLT Tx86: 10626 BPI -- Cmpt III - Alpha/Integrity servers only
TK87	DLT Tx87: 62500 BPI -- Cmpt III - Alpha/Integrity servers only
TK88	DLT Tx88: (Quantum 4000) -- Cmpt IV - Alpha/Integrity servers only
TK89	DLT Tx89: (Quantum 7000) -- Cmpt IV - Alpha/Integrity servers only
QIC	All QIC drives are drive-settable only - Alpha/Integrity servers only
8200	Exa-Byte 8200 - Alpha/Integrity servers only
8500	Exa-Byte 8500 - Alpha/Integrity servers only
DDS1	Digital Data Storage 1 -- 2G - Alpha/Integrity servers only
DDS2	Digital Data Storage 2 -- 4G - Alpha/Integrity servers only
DDS3	Digital Data Storage 3 -- 8-10G - Alpha/Integrity servers only
DDS4	Digital Data Storage 4 - Alpha/Integrity servers only
AIT1	Sony Advanced Intelligent Tape 1 - Alpha/Integrity servers only

Keyword	Meaning
AIT2	Sony Advanced Intelligent Tape 2 - Alpha/Integrity servers only
AIT3	Sony Advanced Intelligent Tape 3 - Alpha/Integrity servers only
AIT4	Sony Advanced Intelligent Tape 4 - Alpha/Integrity servers only
DLT8000	DLT 8000 - Alpha/Integrity servers only
8900	Exabyte 8900 - Alpha/Integrity servers only
SDLT	SuperDLT1 - Alpha/Integrity servers only
SDLT320	SuperDLT320 - Alpha/Integrity servers only

Note that tape density keywords cannot be abbreviated.

When you initialize a tape with the **INITIALIZE** command and do not specify a density, the tape is initialized at the default density for the media and drive you are using (usually the highest density available).

The density of a tape can only be changed if the tape is at beginning-of-tape (BOT). To change the density of a tape that has previously been recorded, the first operation must be a write operation. If the first operation on the tape is a read operation, the magnetic tape is set to the density at which the first record on the tape was recorded, no matter what density is specified with the **/DENSITY** qualifier.

Example

The following command mounts a tape on the MFA0: drive **/FOREIGN** and assigns it the logical name TAPE. The **/DENSITY** qualifier specifies that the tape is to be written at TK87.

```
$ MOUNT/FOREIGN/DENSITY=TK87  MFA0: TAPE
```

/EXTENSION=*n*

Specifies the number of blocks by which disk files are to be extended on the volume unless otherwise specified by an individual command or program request.

The parameter, *n*, specifies a value from 0 to 65,535 to override the value specified when the volume was initialized.

Example

The following command mounts a volume labeled DOC on DKA0, assigns the logical name WORK, and specifies a default block extent of 64 for the files on WORK:

```
$ MOUNT/EXTENSION=64 DKA0: DOC WORK
```

/FOREIGN

Indicates that the volume is not in the standard format used by the OpenVMS operating system.

Use the **/FOREIGN** qualifier when a magnetic tape volume is not in the standard ANSI format, or when a disk volume is not in Files-11 format.

If you mount a volume with the **/FOREIGN** qualifier, the program you use to read the volume must be able to process the labels on the volume, if any. The OpenVMS operating system does not provide an ancillary control process (ACP) to process the volume.

You must mount DOS-1 and RT-11 volumes with the **/FOREIGN** qualifier and process them with the Exchange utility (**EXCHANGE**). See the *OpenVMS Exchange Utility Manual* for more information.

The default protection applied to foreign volumes is RWLP (Read, Write, Logical I/O, Physical I/O) for the system and owner and no access for the group and world. If you also specify **/GROUP**, group members are also given RWLP access. If you specify **/SYSTEM** or **/SHARE**, the group and world are both given RWLP access. Note that the **/GROUP**, **/SYSTEM**, and **/SHARE** qualifiers do not alter the default protection.

If you mount a volume currently in Files-11 format with the **/FOREIGN** qualifier, you must have the user privilege VOLPRO, or your UIC must match the UIC on the volume.

The **/FOREIGN** qualifier is incompatible with the following qualifiers: **/ACCESSED**, **/AUTOMATIC**, **/BIND**, **/CACHE**, **/[NO]CONFIRM**, **/[NO]COPY**, **/EXTENSION**, **/HDR3**, **/INITIALIZE**, **/LABEL**, **/PROCESSOR**, **/QUOTA**, **/REBUILD**, **/SHADOW**, **/OVERRIDE=EXPIRATION**, and **/WINDOWS**.

Examples

The following command mounts a foreign magnetic tape on drive MTA1:

```
$ MOUNT/FOREIGN MTA1: ABCD TAPE
```

The following command mounts an RK07 device as a foreign volume on DMA2 and assigns the default logical name as DISK$SAVEDISK. As a volume that is not file structured, SAVEDISK can be used for sequential-disk BACKUP save operations.

```
$ MOUNT/FOREIGN DMA2: SAVEDISK
```

/GROUP

Makes the volume available to other users with the same group number in their UICs as the user entering the **MOUNT** command.

The logical name for the volume is placed in the group logical name table. You must have the user privilege GRPNAM to use the **/GROUP** qualifier.

Note that if the volume is owned by a group other than yours, access may be denied because of the volume protection.

The **/GROUP** qualifier is not valid for ISO 9660 volume sets.

The **/GROUP** qualifier is incompatible with the **/OVERRIDE=IDENTIFICATION**, **/SHARE**, and **/SYSTEM** qualifiers.

Examples

The following command mounts and makes available on a group basis the volume set consisting of volumes labeled PAYVOL1, PAYVOL2, and PAYVOL3. The logical name PAY is assigned to the set; anyone wanting to access files on these volumes can refer to the set as PAY.

```
$ MOUNT/GROUP DB1:, DB2:, DB3: PAYVOL1,PAYVOL2,PAYVOL3  PAY
```

The following command adds the volume labeled PAYVOL4 to the existing volume set MASTER_PAY. The root volume for the volume set must be on line when you enter this command.

```
$ MOUNT/GROUP/BIND=MASTER_PAY DB4: PAYVOL4
```

/HDR3 (default)
/NOHDR3

Controls whether ANSI standard header label 3 is written on a magnetic tape volume.

By default, header label 3 is written. You can specify the **/NOHDR3** qualifier to write magnetic tapes that are to be used on other systems that do not process HDR3 labels correctly.

Example

In the following example, the **INITIALIZE** and **MOUNT** commands prepare an ANSI-formatted magnetic tape for processing. The **/NOHDR3** qualifier specifies that no HDR3 labels are to be written, thus creating a magnetic tape that can be transported to systems that do not process implementation-dependent labels correctly.

```
$ INITIALIZE  MTA0: ABCD
$ MOUNT/NOHDR3 MTA0: ABCD
```

/INCLUDE *virtual-unit-name*[:] /SHADOW=(*physical-device-name*[:][,...])
/NOINCLUDE *virtual-unit-name*[:] /SHADOW=(*physical-device-name*[:][,...]) (default)

Automatically reconstructs a former shadow set to the way it was before the shadow set was dissolved. This qualifier is applicable only if you have the volume shadowing option. See the *VSI OpenVMS Volume Shadowing Guide* [https://docs.vmssoftware.com/vsi-openvms-volume-shadowing-guide/] for additional information.

The **/INCLUDE** qualifier automatically mounts and restores a shadow set to the way it was before a system failure. Supply the exact virtual-unit name that was used when the shadow set was originally mounted. Use the virtual-unit naming format DSAnnnn:

You must also include the **/SHADOW** qualifier and specify at least one of the disk devices from the original shadow set. Use the standard device-naming format $allocation-class$ddcu[:]. Omit the parentheses if you name only one device.

The **/INCLUDE** qualifier is position independent; it can appear anywhere on the command line.

The default qualifier is **/NOINCLUDE**.

Example

The following example shows how to create a shadow set wherein the software determines automatically the shadow set members that should be mounted. The **/SHADOW** qualifier ensures the correct copy operation for the two shadow set members. In this case, `$1$DUA10` is the more current volume and becomes the source of the copy operation to `$1$DUA11`.

If the shadow set was properly dismounted and no write I/O requests remain outstanding, the shadow set devices are consistent and are added back without the need for a copy or merge operation. Otherwise, Volume Shadowing for OpenVMS automatically performs a copy or merge operation.

```
$ MOUNT/INCLUDE DSA0: /SHADOW=$1$DUA10: SHADOWVOL
%MOUNT-I-MOUNTED, SHADOWVOL mounted on DSA0:
%MOUNT-I-SHDWMEMSUCC, _$1$DUA10: (MEMBER1) is now a valid member ofthe shadow set
%MOUNT-I-SHDWMEMCOPY, _$1$DUA11: (MEMBER2) added to the shadow setwith a copy
 operation
```

/INITIALIZE=CONTINUATION

Specifies that any volume added to the magnetic tape volume set is initialized before you can write to the volume.

Example

The **/INITIALIZE=CONTINUATION** qualifier instructs the **MOUNT** command to assign its own continuation label. In this case, the operator can enter the command **REPLY/BLANK=*n***, and the system assigns a label derived from the original. It uses the label specified in the **MOUNT** command and adds the appropriate number (ABCD02, ABCD03, and so forth).

```
$ MOUNT/INITIALIZE=CONTINUATION MTA0: ABCD
```

/LABEL (default)
/NOLABEL

Indicates that the volume is in the standard format used by the OpenVMS operating system; that is, a magnetic tape volume is in the standard ANSI format, or a disk volume is in Files-11 format.

The default is **/LABEL**.

Note that **/NOLABEL** is equivalent to **/FOREIGN**; they both set the FOREIGN flag.

Example

The following command mounts an ANSI-labeled magnetic tape on MFA1 and assigns the default logical name as TAPE$TAPE.

```
$ MOUNT/LABEL MFA1: TAPE
```

/MEDIA_FORMAT=CDROM

Mounts a volume assuming the media to be ISO 9660 (or High Sierra) formatted.

The **/MEDIA_FORMAT=CDROM** qualifier instructs the mount subsystem to attempt to mount a volume assuming the media to be ISO 9660 (or High Sierra) formatted.

Note

This qualifier specifies a CD-ROM mount (ISO 9660 or High Sierra). Specify this qualifier when a volume is known to be in either ISO 9660 or High Sierra CD-ROM format.

The **MOUNT** command attempts to read a CD-ROM in Files-11 ODS-2 format by default. This qualifier prevents the **MOUNT** command from attempting a Files-11 ODS-2 mount sequence.

Because it is possible to record parts of a CD-ROM in Files-11 ODS-2 and other parts in ISO 9660 format, this qualifier can be used to specify a CD-ROM mount (ISO 9660 or High Sierra).

/MEDIA_FORMAT= [NO]COMPACTION

Enables and controls data compaction and data record blocking on tape drives that support data compaction.

The **/MEDIA_FORMAT** qualifier allows you to mount a tape and enable data compaction and record blocking on a tape drive that supports data compaction. Data compaction and record blocking increase the amount of data that can be stored on a single tape.

Records can either be compacted and blocked, or they can be recorded in the same way that they would be recorded on a non compacting tape drive. Note that for compacting tape drives, once data compaction or non compaction has been selected for a given tape, that status applies to the entire tape.

The **/MEDIA_FORMAT=[NO]COMPACTION** qualifier is incompatible with the **/DENSITY** qualifier.

For Files-11 tapes, when you enable data compaction, caching is automatically enabled.

Note

The **/MEDIA_FORMAT=[NO]COMPACTION** qualifier is meaningful only for foreign mounts.

The **/MEDIA_FORMAT=[NO]COMPACTION** qualifier has no effect on a Files-11 tape. The compaction state of a Files-11 tape is determined by the state established when the tape is initialized.

Examples

The following command performs a foreign mount of a tape with data compaction and record blocking enabled and assigns the logical name BOOKS to the tape:

```
$ MOUNT/FOREIGN/MEDIA_FORMAT=COMPACTION MUA0: BOOKS
```

The following **MOUNT** command attempts a Files-11 mount of a tape labeled BOOKS with data compaction and record blocking enabled. Because the tape was initialized with compaction disabled, the **MOUNT** qualifier **/MEDIA_FORMAT=COMPACTION** has no effect.

```
$ INIT/MEDIA_FORMAT=NOCOMPACTION MUA0: BOOKS
$ MOUNT/MEDIA_FORMAT=COMPACTION MUA0: BOOKS
```

/MESSAGE (default)
/NOMESSAGE

Causes mount request messages to be sent to your current SYS$OUTPUT device.

If you specify **/NOMESSAGE** during an operator-assisted mount, messages are not output to SYS$OUTPUT; the operator sees them, however, provided an operator terminal is enabled.

Example

In this example, an RL02 device labeled SLIP is mounted on drive DLA0 and is assigned the logical name DISC. The **/NOMESSAGE** qualifier disables the broadcast of mount request messages to the user terminal.

```
$ MOUNT/NOMESSAGE DLA0: SLIP DISC
```

/MOUNT_VERIFICATION (default)
/NOMOUNT_VERIFICATION

Specifies that the device is a candidate for mount verification.

The **/MOUNT_VERIFICATION** qualifier affects the following media:

- Files-11 Structure Level 2 or 5 disks (mount verification is not supported for foreign-mounted disks)

- ISO 9660 and High Sierra CD-ROMs
- Foreign and ANSI-labeled magnetic tape volumes

Example

The following command mounts an HSG80 Fibre Channel disk device labeled FILES and assigns the logical name WORK. The **/CACHE** qualifier disables extent caching, file identification caching, quota caching, data caching, and write back caching; the **/NOMOUNT_VERIFICATION** qualifier disables mount verification.

```
$ MOUNT/CACHE=(NOEXTENT,NOFILE_ID,NOQUOTA,WRITETHROUGH) -
_$ /NOMOUNT_VERIFICATION  $1$DGA0: FILES WORK
%MOUNT-I-MOUNTED, FILES          mounted on _$1$DGA0: (NODE)
```

/MULTI_VOLUME
/NOMULTI_VOLUME (default)

For foreign or unlabeled magnetic tape volumes, determines whether you override MOUNT volume-access checks.

Use **/MULTI_VOLUME** to override access checks on volumes that do not contain labels that MOUNT can interpret. If you have software produced before OpenVMS Version 5.0 that processes multiple-volume, foreign-mounted tape volumes without specifically mounting and dismounting each reel, you may now need to mount the first volume with the **/MULTI_VOLUME** qualifier.

Use this qualifier when a utility that supports multiple-volume, foreign-mounted magnetic tape sets needs to process subsequent volumes, and these volumes do not contain labels that the OpenVMS Mount command can interpret.

By default, all tape volumes are subject to the complete access checks of the OpenVMS Mount command (**MOUNT**). Some user-written and vendor-supplied utilities used prior to OpenVMS Version 5.0 may mount only the first tape in a foreign tape set. To make these utilities compatible with more recent versions of OpenVMS, alter them to perform explicit calls to the $MOUNT and $DISMOU system services for each reel in the set. As an alternative, you can now mount the magnetic tape sets to be used by these utilities with the **/MULTI_VOLUME** qualifier.

You must specify the **/FOREIGN** qualifier with the **/MULTI_VOLUME** qualifier and you must have the user privilege VOLPRO. The default is **/NOMULTI_VOLUME**.

Note

The OpenVMS Backup utility (**BACKUP**) explicitly calls the $MOUNT and $DISMOU system services on each reel of a foreign-mounted magnetic tape set. For additional information, see the section on multivolume save sets and BACKUP in the *VSI OpenVMS System Management Utilities Reference Manual, Volume 1: A-L* [https://docs.vmssoftware.com/vsi-openvms-system-management-utilities-reference-manual-volume-i-a-l/#BKU_U].

Example

The following command mounts a tape volume set. MOUNT performs an access check on the first volume in the set and proceeds without checks to subsequent reels as they are needed for processing.

```
$ MOUNT/FOREIGN/MULTI_VOLUME MUA0:
```

/OVERRIDE=(*keyword*[,...])

Inhibits one or more protection checks that the **MOUNT** command performs.

You need the user privileges OPER and VOLPRO to specify **/OVERRIDE=(ACCESSIBILITY, EXPIRATION)** along with the **/FOREIGN** qualifier; otherwise, the magnetic tape is not read.

If you specify more than one keyword, separate them with commas and enclose the list in parentheses.

The following table lists the keywords for this qualifier:

Keyword	Description
ACCESSIBILITY	For magnetic tapes only. If the installation allows, this keyword overrides any character in the Accessibility Field of the volume. The necessity of this keyword 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, the OpenVMS operating system provides a routine that checks this field in the following manner: ● If the magnetic tape was created on a version of OpenVMS that conforms to Version 3 of ANSI, then you must use this keyword to override any character other than an ASCII space. ● If an OpenVMS protection is specified and the magnetic tape conforms to an ANSI standard that is higher than Version 3, then you must use this keyword to override any character other than an ASCII 1. To use the ACCESSIBILITY keyword, you must have the user privilege VOLPRO or own the volume.
EXPIRATION	For magnetic tapes only. Allows you to override the expiration dates of a volume and its files. Use this keyword when the expiration date in the first file header label of any file that you want to overwrite has not been reached. You must have the user privilege VOLPRO or your UIC must match the UIC written on the volume.
IDENTIFICATION	Overrides processing of the volume identifier in the volume label. Use this keyword to mount a volume for which you do not know the label, or for an ISO 9660 volume whose label is not unique in the first 12 characters. Only the volume identifier field is overridden. Volume protection, if any, is preserved. The volume must be mounted **/NOSHARE** (either explicitly or by default). The **/OVERRIDE=IDENTIFICATION** qualifier is incompatible with the **/GROUP** and **/SYSTEM** qualifiers.
LIMITED_SEARCH	Allows the **MOUNT** command to search an entire device for a home block, if a home block is not found at the expected

Keyword	Description
	location. By default, the search or a home block is limited to avoid excessive search times if no valid home block is present.
LOCK	Directs MOUNT not to write-lock the volume as a consequence of certain errors encountered while mounting it. Use this keyword when you are mounting a damaged volume to be repaired using the **ANALYZE/DISK_STRUCTURE** command. You must have VOLPRO privilege or own the volume to use the LOCK keyword.
NO_FORCED_ERROR	Directs the **MOUNT** command to proceed with shadowing, even though the device or controller does not support forced error handling. Using unsupported SCSI disks can cause members to be removed from a shadow set if certain error conditions arise that cannot be corrected, because some SCSI disks do not implement **READL** and **WRITEL** commands that support disk bad block repair.
OWNER_IDENTIFIER	For magnetic tapes only. Overrides the processing of the owner identifier field. Use this keyword to interchange protected magnetic tapes between OpenVMS and other operating systems.
SECURITY	Allows you to continue mounting a volume if an error is returned because the volume has an invalid SECURITY.SYS file. You must have the user privilege VOLPRO or own the volume to use this keyword.
SETID	For magnetic tapes only. Prevents MOUNT from checking the file-set identifier in the first file header label of the first file on a continuation volume. Use this keyword only for ANSI-labeled volumes on which the file-set identifier of the first file on a continuation volume differs from the file-set identifier of the first file of the first volume that was mounted.
SHADOW_MEMBERSHIP	Allows you to override the write protection of former shadow set members. Applicable only if you have the volume shadowing option. See the *VSI OpenVMS Volume Shadowing Guide* [https://docs.vmssoftware.com/vsi-openvms-volume-shadowing-guide/]. When you mount a volume with this qualifier, the volume shadowing generation number is erased. If you attempt to remount the volume in a shadow set, the volume is considered an unrelated volume and receives a full copy operation from a current shadow set member.

The following command overrides the volume identification field, thus mounting a magnetic tape on `MFA0` without a label specification:

```
$ MOUNT/OVERRIDE=IDENTIFICATION MFA0:
```

/OWNER_UIC=*uic*

Requests that the specified UIC be assigned ownership of the volume while it is mounted, overriding the ownership recorded on the volume. If you are mounting a volume using the **/FOREIGN** qualifier, requests an owner UIC other than your current UIC.

The parameter, *uic*, specifies the user identification code (UIC) in the following format:

[group,member]

You must use brackets in the UIC specification. The group number is an octal number in the range 0 to 37776;the member number is an octal number in the range 0 to 177776.

To use the **/OWNER_UIC** qualifier for a Files-11 volume, you must have the user privilege VOLPRO, or your UIC must match the UIC written on the volume.

Example

The following command mounts a disk device labeled `WORK` on `DRA3` and assigns an owner UIC of `[016,360]`:

```
$ MOUNT/OWNER_UIC=[016,360] DRA3: WORK
```

/POLICY=[NO]MINICOPY[= (OPTIONAL)], REQUIRE_MEMBERS, [NO]VERIFY_LABEL

Controls the setup and use of shadow sets. For more information about volume shadowing, see the *VSI OpenVMS Volume Shadowing Guide* [https://docs.vmssoftware.com/vsi-openvms-volume-shadowing-guide/]. The following table lists the keywords for this qualifier:

Keyword	Description
[NO]MINICOPY [=OPTIONAL] (Alpha/Integrity servers only)	Controls the setup and use of the shadowing minicopy function. **Requires LOG_IO (logical I/O) privilege to create bitmaps.** The meaning of the keyword [NO]MINICOPY[=OPTIONAL] for the **MOUNT/POLICY** qualifier depends on the status of the shadow set, as follows: 1. If the shadow set is not mounted, either on a standalone system or on any cluster member, and MINICOPY=OPTIONAL is specified, the shadow set is mounted and a write bitmap is created. The write bitmap enables a shadowing minicopy operation. You must specify **MOUNT/POLICY=MINICOPY[=OPTIONAL]** on the initial mount of a shadow set, either on a standalone system or in a cluster, to enable the shadowing minicopy operation. The OPTIONAL keyword allows the mount to continue, even if the system was unable to start the write bitmap. Likely reasons for the bitmap to fail to start properly include an improperly dismounted shadow set, a shadow set that requires a merge operation, and various resource problems. If the OPTIONAL keyword is omitted and the system is unable to start the write bitmap, the shadow set will not be mounted. If you specify the **/POLICY=MINICOPY=OPTIONAL** qualifier and the shadow set was already mounted on another node in the cluster without the **/POLICY=MINICOPY[=OPTIONAL]**, the **MOUNT** command succeeds but a write bitmap is not created. If NOMINICOPY is specified, the shadow set is mounted but a write bitmap is not created.

Keyword	Description
	2. If a former member of the shadow set is returned to the shadow set, which has minicopy enabled, then a minicopy is started instead of a full copy. This is the default behavior and will occur even if you omit **/POLICY=MINICOPY[=OPTIONAL]**. If a minicopy is successfully started and then fails for some reasons, a full copy is performed. If a minicopy cannot be started and the keyword OPTIONAL was omitted, the mount will fail. If NOMINICOPY is specified, then no minicopy is performed, even if one is possible.
REQUIRE_MEMBERS	Controls whether every physical device specified with the **/SHADOW** qualifier must be accessible when the **MOUNT** command is issued in order for the **MOUNT** command to take effect. The proposed members are either specified in the command line or found on the disk by means of the **/INCLUDE** qualifier. The behavior, without this qualifier, is that if one or more members is not accessible for any reason (such as a connectivity failure), then the virtual unit will be created with the members that are accessible. This option is especially useful in the recovery of disaster-tolerant clusters because it ensures that the correct membership is selected after an event.
[NO]VERIFY_LABEL	Require that any member that is going to be added to the shadow set must have a volume label of 'SCRATCH_DISK'. This will help insure that the wrong disk is not added to a shadow set by mistake. If VERIFY_LABEL is going to be used, then the disk that is going to be added to the set must be either initialized with the label 'SCRATCH_DISK' or a **SET VOLUME/LABEL** must be performed. The default behavior is NOVERIFY_LABEL, which indicates that the volume label of the copy targets will not be checked.

/PROCESSOR=*keyword*

For magnetic tapes and Files-11 Structure Level 1 disks, requests that the **MOUNT** command associate an ancillary control process (ACP) to process the volume. The **/PROCESSOR** qualifier causes MOUNT to override the default manner in which ACPs are associated with devices.

For Files-11 Structure Levels 2 and 5 disks, controls block cache allocation.

The following table lists the keywords for this qualifier:

Keyword	Description
UNIQUE	Creates a new process to execute the default ancillary control process (ACP) image supporting the magnetic tape, Files-11 ODS-1, ISO 9660, or High Sierra formatted media being mounted.

Keyword	Description
	For Files-11 Structure Levels 2 and 5 disks, allocates a separate block cache.
SAME:device	Uses an existing process that is executing the same ACP image supporting the magnetic tape, Files-11 ODS-1, ISO 9660, or High Sierra formatted media being mounted. For Files-11 Structure Levels 2 and 5 disks, takes the block cache allocation from the specified device.
file-spec	Creates a new process to execute the ACP image specified by the file specification (for example, a modified or a user-written ACP). You cannot use wildcard characters, or node and directory names in the file specification. To use this keyword, you need CMKRNL and OPER privileges. You must have the operator user privilege OPER to use the **/PROCESSOR** qualifier.

Example

The following command directs MOUNT to mount a magnetic tape on `MFA0` using the same ACP process currently associated with `MTA1`:

```
$ MOUNT/PROCESSOR=SAME:MTA1: MFA0:
```

/PROTECTION=*keyword*

Specifies the protection code to be assigned to the volume.

The following table describes the keywords for this qualifier:

Keyword	Description
protection code	Specifies the protection code according to the standard syntax rules for specifying user protection (that is, system/owner/group/world). If you omit a protection category, that category of user is denied all access. If you do not specify a protection code, the default is the protection that was assigned to the volume when it was initialized.
XAR	Enables enforcement of the extended record attribute (XAR) access controls. For more information about XAR, see the *VSI OpenVMS Record Management Services Reference Manual* [https://docs.vmssoftware.com/vsi-openvms-record-management-services-reference-manual/].
DSI	Enables XAR permissions Owner and Group for XARs containing Digital System Identifiers (DSI). For more information, see the *VSI OpenVMS Record Management Services Reference Manual* [https://docs.vmssoftware.com/vsi-openvms-record-management-services-reference-manual/].

If you specify the **/PROTECTION** qualifier when you mount a volume with the **/SYSTEM** or **/GROUP** qualifier, the specified protection code overrides any access rights implied by the other qualifiers.

If you specify the **/FOREIGN** qualifier, the execute (E) or create (C) and delete (D) access codes are synonyms for logical I/O (L) and physical I/O (P). You can, however, specify the access codes physical I/O (P) or logical I/O (L), or both, to restrict the nature of input/output operations that different user categories can perform.

To use the **/PROTECTION** qualifier on a Files-11 volume, you must have the user privilege VOLPRO or your UIC must match the UIC written on the volume.

Example

The following command mounts a device labeled WORKDISK on DKA1 and assigns a protection code. Access to the volume will be read, write, and create for system users; read, write, create, and delete for owner; read and create for group users; and read-only for users in the world category.

```
$ MOUNT/PROTECTION=(SYSTEM:RWE,O:RWED,G:RE,W:R) DKA1: WORKDISK
```

/QUOTA (default)
/NOQUOTA

Controls whether quotas are to be enforced on the specified disk volume.

The default is **/QUOTA**, which enforces the quotas for each user. The **/NOQUOTA** qualifier inhibits this checking. To specify the **/QUOTA** qualifier, you must have the user privilege VOLPRO or your UIC must match the UIC written on the volume.

Example

The following command specifies that the disk volume labeled WORK on DRA3 has an owner UIC of [016,360] and no quotas enforced:

```
$ MOUNT/OWNER_UIC=[016,360]/NOQUOTA DRA3: WORK
```

/REBUILD (default)
/NOREBUILD

Controls whether or not MOUNT performs a rebuild operation on a disk volume.

If a disk volume is improperly dismounted (such as during a system failure), you must rebuild it to recover any caching limits that were enabled on the volume at the time of the dismount. By default, MOUNT attempts the rebuild. For a successful rebuild operation that includes reclaiming all of the available free space, you must mount *all* of the volume set members.

The rebuild may consume a considerable amount of time, depending on the number of files on the volume and, if quotas are in use, on the number of different file owners.

The following caches may have been in effect on the volume before it was dismounted:

- Preallocated free space (EXTENT cache)
- Preallocated file numbers (FILE_ID cache)
- Disk quota usage caching (QUOTA cache)

If caching was in effect for preallocated free space or file numbers, the rebuild time is directly proportional to the greatest number of files that ever existed on the volume at one time. If disk quota

caching was in effect, you can expect additional time that is proportional to the square of the number of entries in the disk quota file.

If none of these items were in effect, the rebuild is not necessary and does not occur.

If you use the **/NOREBUILD** qualifier, devices can be returned to active use immediately. You can then perform the rebuild later with the DCL command **SET VOLUME/REBUILD**.

For information about how to rebuild the system disk, see the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/].

Examples

In this example, the volume WORKDISK is mounted on NODE$DKA2. Because the volume is found to have been improperly dismounted and the **/REBUILD** qualifier is in effect, MOUNT displays a message and proceeds to rebuild the volume.

```
$ MOUNT/REBUILD NODE$DKA2: WORKDISK
%MOUNT-I-MOUNTED, WORKDISK         mounted on _NODE$DKA2:
%MOUNT-I-REBUILD, volume was improperly dismounted; rebuild in progress
```

In this example, the volume WORKDISK is found to have been improperly dismounted, but because the **/NOREBUILD** qualifier is specified, a rebuild is not performed. Instead, MOUNT displays a message to inform you that the rebuild is needed, and proceeds to make WORKDISK available for use as is. You can rebuild the volume later with the DCL command **SET VOLUME/REBUILD**.

```
$ MOUNT/NOREBUILD NODE$DKA2: WORKDISK
%MOUNT-I-MOUNTED, WORKDISK          mounted on _NODE$DKA2:
%MOUNT-I-REBLDREQD, rebuild not performed; some free space unavailable;
 diskquota usage stale
```

/RECORDSIZE=*n*

Specifies the number of characters in each record of a magnetic tape volume.

The parameter, *n*, specifies the block size in the range 20 to 65,532 bytes if you are using OpenVMS RMS, or 18 to 65,534 bytes if you are not using OpenVMS RMS.

You typically use this qualifier with the **/FOREIGN** and **/BLOCKSIZE** qualifiers to read or write fixed-length records on a block-structured device. In this case, the record size must be less than or equal to the block size specified or used by default.

Use the **/RECORDSIZE** qualifier when mounting magnetic tapes without HDR2 labels (such as RT-11 magnetic tapes) to provide OpenVMS RMS with default values for the maximum record size.

Example

In the following example, the magnetic tape is mounted on MTA0 with a default block size and record size of 512 characters:

```
$ MOUNT/FOREIGN/BLOCKSIZE=512/RECORDSIZE=512 MTA0:
```

/SHADOW

Binds up to three physical devices into a shadow set represented by the virtual unit named in the command. This qualifier is applicable only if you have the volume shadowing option. See the

VSI OpenVMS Volume Shadowing Guide [https://docs.vmssoftware.com/vsi-openvms-volume-shadowing-guide/] for additional information.

The format of this qualifier is:

```
(virtual-unit-name[:] /SHADOW=(physical-device-name[:][,...]))
```

This qualifier indicates that you are mounting a shadow set including the physical devices and the virtual unit that represents them to the system. This qualifier instructs MOUNT to expect a virtual unit name as the *device-name* parameter. Place the **/SHADOW** qualifier after the *virtual-unit-name parameter*.

Use the virtual unit naming format DSA *n*, where *n* is a unique number from 0 to 9999. For the *physical-device-name*, use the standard device-naming format $allocation-class$ddcu[:].

Examples

The following example shows how to create a shadow set wherein the software determines automatically the correct copy operation for the two shadow set members. In this case, `$1$DUA10` is the more current volume and becomes the source of the copy operation to `$1$DUA11`.

```
$ MOUNT DSA0: /SHADOW=($1$DUA10:,$1$DUA11:) SHADOWVOL
%MOUNT-I-MOUNTED, SHADOWVOL mounted on DSA0:
%MOUNT-I-SHDWMEMSUCC, _$1$DUA10: (MEMBER1) is now a valid member ofthe
 shadow set
%MOUNT-I-SHDWMEMCOPY, _$1$DUA11: (MEMBER2) added to the shadow setwith a
 copy operation
```

The following command creates a volume set with the logical name `TEST3013`. The volume set `TEST3013` is not shadowed. However, each element of the volume set (`TEST3011` and `TEST3012`) is a shadow set, providing redundancy for the volume set as a whole.

```
$ MOUNT/BIND=TEST3013 DSA3011/SHADOW=($1$DUA402:,$1$DUA403:),
DSA3012/SHADOW=($1$DUA404:,$1$DUA405:) TEST3011,TEST3012 TEST3013
```

/SHARE
/NOSHARE

Specifies, for a disk volume, that the volume is shareable.

If another user has already mounted the volume shareable, and you request it to be mounted with the **/SHARE** qualifier, any other qualifiers you enter are ignored.

By default, a volume is not shareable, and the **MOUNT** command allocates the device on which it is mounted.

If you previously allocated the device and specify the **/SHARE** qualifier, the **MOUNT** command deallocates the device so that other users can access it.

The **/SHARE** qualifier is incompatible with the **/GROUP** and **/SYSTEM** qualifiers.

Example

The following command mounts the device labeled `SLIP` on `DLA0`, disables broadcasting of MOUNT messages, specifies that the volume is shareable, and assigns the logical name `DISC`:

```
$ MOUNT/NOMESSAGE/SHARE DLA0: SLIP DISC
```

/SUBSYSTEM
/NOSUBSYSTEM

Enables protected subsystems and the processing of subsystem ACEs. Requires the SECURITY privilege.

By default, the disk from which you boot has **/SUBSYSTEM** enabled but other disks do not. For further details on subsystems, see the relevant section in the *VSI OpenVMS Guide to System Security* [https://docs.vmssoftware.com/vsi-openvms-guide-to-system-security/#PROTECT_SUBSYSTEM].

Example

The following command mounts the volume labeled SLIP on DUA1 with mount messages disabled. Subsystems on the volume are accessible. MOUNT also assigns the logical name SACH.

```
$ MOUNT/NOMESSAGE/SUBSYSTEM DUA1: SLIP SACH
```

/SYSTEM

Makes the volume public; that is, available to all users of the system, as long as the UIC-based volume protection allows them access.

The logical name for the device is placed in the system logical name table. You must have the user privilege SYSNAM to use the **/SYSTEM** qualifier.

When you mount a volume with the **/SYSTEM** qualifier in a VMS cluster system, you must use a volume label that is unique clusterwide, even if the specified volume is not mounted clusterwide.

The **/SYSTEM** qualifier is incompatible with the **/GROUP**, **/OVERRIDE=IDENTIFICATION**, and **/SHARE** qualifiers.

Examples

The following command mounts the volume labeled SLIP on DUA1 with mount messages disabled. The volume is made available systemwide. MOUNT also assigns the logical name SACH.

```
$ MOUNT/NOMESSAGE/SYSTEM DUA1: SLIP SACH
```

The following command creates the volume set named MASTER_PAY consisting of the initialized volumes labeled PAYVOL1, PAYVOL2, and PAYVOL3. These volumes are mounted physically on the devices named DB1, DB2, and DB3, respectively. The volume PAYVOL1 is the root volume of the set.

The volumes are mounted as system volumes to make them available to all users.

```
$ MOUNT/SYSTEM/BIND=MASTER_PAY -
_$ DB1:,DB2:,DB3:     PAYVOL1,PAYVOL2,PAYVOL3
```

/UCS_SEQUENCE=*escape_sequence*

Supplies the escape sequence to select the coded graphic character set, a requirement when mounting an ISO 9660 volume for one of the Supplementary Volume Descriptors (SVDs).

The parameter, *escape_sequence*, is a character sequence defined by the vendor who mastered the CD-ROM and is unique to the vendor's character set conversion tables.

Use the **/UCS_SEQUENCE** qualifier when mounting an ISO 9660 CD-ROM that contains non-ASCII character sets on OpenVMS.

An ISO 9660 volume may contain an SVD that specifies a graphic character set. This graphic character, when selected at mount time, is used as default character set when displaying a volume's directories and file names.

The **/UCS_SEQUENCE** qualifier defines the escape sequence to select the coded graphic character set.

All ISO 9660 volumes contain a Primary Volume Descriptor (PVD) that uses ASCII (ISO 646-IRV) as the character set. Both ISO 9660 and OpenVMS file naming conventions use the same subset of ASCII characters when displaying a volume's directories and file names.

/UNDEFINED_FAT=*record-format*:[*record-attributes*:][*record-size*]

Establishes default file attributes to be used for records on ISO 9660 media for which no record format has been specified.

The following table describes the parameters:

Parameter	Description
record-format	Specifies the format for all records in a file: FIXED, VARIABLE, STREAM, STREAM_LF, STREAM_CR, LSB_VARIABLE, or MSB_VARIABLE. For a description of these record formats, see the discussion of the RMS field FAB$B_RFM in the *VSI OpenVMS Record Management Services Reference Manual* [https://docs.vmssoftware.com/vsi-openvms-record-management-services-reference-manual/#RMS_RECORD_FORMAT_FIELD].
record-attributes	Specifies the attributes for all records in a file: NONE, CR, FTN, PRN, NOBKS. Applies only to non-STREAM record formats. For a description of these record attributes, see the discussion of the RMS field FAB $B_RAT in the *VSI OpenVMS Record Management Services Reference Manual* [https://docs.vmssoftware.com/vsi-openvms-record-management-services-reference-manual/#RMS_RECORD_ATTRIBUTE_FIELD].
record-size	Specifies the maximum record size for all records in a file: 0 to 32767. Applies only to FIXED or STREAM record formats. For a description of possible RMS record sizes, see the discussion of the RMS field FAB $W_MRS in the *VSI OpenVMS Record Management Services Reference Manual* [https://docs.vmssoftware.com/vsi-openvms-record-management-services-reference-manual/#_1344_FABW_MRSFIELD].

ISO 9660 media can be mastered from platforms that do not support semantics of files containing predefined record formats. The **/UNDEFINED_FAT** qualifier establishes default file attributes to be used for records on ISO 9660 media for which no record format has been specified.

The **/UNDEFINED_FAT** qualifier is valid only in conjunction with the **/MEDIA_FORMAT=CDROM** qualifier.

This qualifier temporarily overrides *all* undefined file types, replacing them with selectable record formats having selectable record attributes and selectable record sizes as shown in the following illustrations:

record_format

1. FIXED:*record-attributes*[, …]:*record-size*

2. VARIABLE:*record-attributes*[, …]
3. STREAM:*record-size*
4. STREAM_LF:*record-size*
5. STREAM_CR:*record-size*
6. LSB_VARIABLE:*record-attributes*[, …]

record_attributes

1. NONE - None
2. CR - Carriage_return
3. FTN - Fortran
4. PRN - Print
5. NOBKS - No-Block-Span

record_size

1. 1 to 32767

Example

In the following example, the volume labeled OFFENS is mounted on DKA1 and all files on the volume are defined to be fixed length, carriage return, and 80 bytes in length. MOUNT also assigns the logical name STRAT.

```
$ MOUNT/MEDIA_FORMAT=CDROM/UNDEFINED_FAT=(FIXED:CR:80) DKA1: OFFENS
 STRAT
```

/UNLOAD (default)
/NOUNLOAD

Controls whether or not the disk or magnetic tape volume or volumes specified in the **MOUNT** command are unloaded when they are dismounted.

Example

In the following example, the volume labeled OFFENS is mounted on DKA1 with the **/NOUNLOAD** qualifier so that it can be dismounted without being physically unloaded. MOUNT also assigns the logical name STRAT.

```
$ MOUNT/NOUNLOAD DKA1: OFFENS STRAT
```

/WINDOWS=*n*

Specifies the number of mapping pointers to be allocated for file windows.

The parameter, *n*, specifies a value from 7 to 80 that overrides the default value specified when the volume was initialized.

When a file is opened, the file system uses the mapping pointers to access data in the file. Use **MOUNT/WINDOWS** to override the default value specified when the volume was initialized. If no value was specified at volume initialization, the default number of mapping pointers is 7.

You must have the operator user privilege (OPER) to use the **/WINDOWS** qualifier.

Example

The following command makes the volume labeled GONWITH on DKA2 available systemwide and assigns the logical name THE_WINDOW. You override the default number of mapping pointers by specifying a value of 25 for the **/WINDOWS** qualifier.

```
$ MOUNT/SYSTEM/WINDOWS=25 DKA2: GONWITH THE_WINDOW
```

/WRITE (default)
/NOWRITE

Controls whether the volume can be written.

By default, a volume is considered read/write when it is mounted. You can specify **/NOWRITE** to provide read-only access to protect files. This is equivalent to write-locking the device.

For host-based volume shadowing devices, there are other considerations. See the *VSI OpenVMS Volume Shadowing Guide* [https://docs.vmssoftware.com/vsi-openvms-volume-shadowing-guide/] manual for more information.

Example

The following command mounts a volume labeled BOOKS on NODE$DKA1 and then proceeds to mount it on each node in the existing OpenVMS Cluster. The **/NOWRITE** qualifier makes the volume available for read-only access.

```
$ MOUNT/CLUSTER/NOWRITE NODE$DKA1: BOOKS
```

Examples

For examples 1 and 2, operator assistance is not required, assuming the volumes are in the drives. Examples 3, 4, 5, and 6 describe operator-assisted mounts. Examples 7 and 8 describe mounting ISO 9660 CD-ROM volume sets, example 9 makes subsystems on a volume accessible, and example 10 demonstrates mounting a shadow set.

1.
```
$ MOUNT MTA0: MATH06 STAT_TAPE
%MOUNT-I-MOUNTED, MATH06 mounted on _MTA0:
$ COPY   ST061178.DAT   STAT_TAPE:
```

   This **MOUNT** command requests the magnetic tape whose volume label is MATH06 to be mounted on the device MTA0 and assigns the logical name STAT_TAPE to the volume.

   Subsequently, the **COPY** command copies the disk file ST061178.DAT to the magnetic tape.

2.
```
$ ALLOCATE DM:
%DCL-I-ALLOC, _DMB2:  allocated
$ MOUNT DMB2:  TEST_FILES
%MOUNT-I-MOUNTED, TEST_FILES mounted on _DMB2:
```

   This **ALLOCATE** command requests an available RK06/RK07 device. After the response from the **ALLOCATE** command, the physical volume can be placed on the allocated device. Then, the **MOUNT** command mounts the volume.

3.
```
$ MOUNT DM:  TEST_FILES
```

```
%MOUNT-I-OPRQST, Please mount volume TEST_FILES in device _DMB2:
%MOUNT-I-MOUNTED, TEST_FILES mounted on _DMB2:
```

This example achieves the same result as the series of commands in the preceding example. The **MOUNT** command requests an available RK06/RK07 device or the volume labeled TEST_FILES. After the volume is physically mounted in the device named in the response from MOUNT, the system completes the operation. Note that the device is automatically allocated by MOUNT.

4.
```
$ MOUNT DYA1:  TESTSYS
%MOUNT-I-OPRQST, Please mount volume TESTSYS in device DYA1:
Ctrl/Y
$ EXIT
%MOUNT-I-OPRQSTCAN, operator request canceled
```

This **MOUNT** command requests the operator to mount the volume TESTSYS on the device DYA1. In this example, the user cancels the mount by pressing **Ctrl/Y**. Notice that the image must exit before the mount request is actually canceled. Here, the **EXIT** command causes the image to exit. However, any command that is not performed within the command interpreter causes the current image to exit.

5.
```
$ MOUNT DYA1:  TESTSYS
%MOUNT-I-OPRQST, Device _DYA1: is not available for mounting.
%MOUNT-I-OPRQSTCAN, operator request canceled
%MOUNT-I-OPRQST, Please mount volume TESTSYS in device _DYA1:
%MOUNT-I-MOUNTED, TESTSYS    mounted on _DYA1:
%MOUNT-I-OPRQSTDON, operator request canceled - mount
completed successfully
```

This **MOUNT** command requests the operator to mount the volume TESTSYS on the device DYA1. Because DYA1 is allocated to another user, the device cannot be mounted. In this case, the user can wait for the device to become available, redirect the mount to another device, or abort the mount. Here, the user remains in operator-assisted mount waiting for the process that is using the device to deallocate it.

At this point, because the device is available but no volume is mounted, the original mount request is canceled, and a new request to mount TESTSYS is issued. Finally, the operator places the volume in the drive and lets MOUNT retry the mount. When the mount completes, the request is canceled.

6.
```
$ MOUNT DYA1:  TESTSYS/COMMENT="Is there an operator around?"
%MOUNT-I-OPRQST, Please mount volume TESTSYS in device _DYA1:
Is there an operator around?
%MOUNT-I-NOOPR, no operator available to service request
  .
  .
  .
%MOUNT-I-MOUNTED, TESTSYS    mounted on _DYA1:
%MOUNT-I-OPRQSTDON, operator request canceled - mount
completed successfully
```

This **MOUNT** command requests the operator to mount the volume TESTSYS on the device DYA1. In this example, no operator is available to service the request. At this point, the user can abort the mount by pressing **Ctrl/Y**, or wait for an operator. Here, the user waited, and an operator eventually became available to service the request.

7.
```
$ MOUNT/SYSTEM/MEDIA=CDROM $1$DKA1 USER
%MOUNT-I-CDROM_ISO, USER:VMS_ONLINE_DOCUMENTATION (1 of 4) ,
mounted on _$1$DKA1: (CDROM)
$ MOUNT/SYSTEM/MEDIA=CDROM $1$DKA2 PROGRAMMING_1
%MOUNT-I-CDROM_ISO, PROGRAMMING_1:VMS_ONLINE_DOCUMENTATION (2 of 4) ,
mounted on  _$1$DKA2: (CDROM)
```

```
$  MOUNT/SYSTEM/MEDIA=CDROM $1$DKA3 PROGRAMMING_2
%MOUNT-I-CDROM_ISO, PROGRAMMING_2:VMS_ONLINE_DOCUMENTATION (3 of 4) ,
mounted on  _$1$DKA3: (CDROM)
MOUNT/SYSTEM/MEDIA=CDROM $1$DKA4 MANAGEMENT
%MOUNT-I-CDROM_ISO, MANAGEMENT:VMS_ONLINE_DOCUMENTATION (4 of 4) ,
mounted on  _$1$DKA4: (CDROM)
```

These commands mount each member of a four-member ISO 9660 volume set whose volume-set name is VMS_ONLINE_DOCUMENTATION.

8.
```
$ MOUNT/SYSTEM/MEDIA=CDROM $1$DKA1,$1$DKA2,$1$DKA3,$1$DKA4
USER,PROGRAMMING_1,PROGRAMMING_2,MANAGEMENT
%MOUNT-I-CDROM_ISO, USER:VMS_ONLINE_DOCUMENTATION (1 of 4) , mounted on
_$1$DKA1: (CDROM)
%MOUNT-I-CDROM_ISO, PROGRAMMING_1:VMS_ONLINE_DOCUMENTATION (2 of 4) ,
mounted on  _$1$DKA2: (CDROM)
%MOUNT-I-CDROM_ISO, PROGRAMMING_2:VMS_ONLINE_DOCUMENTATION (3 of 4) ,
mounted on  _$1$DKA3: (CDROM)
%MOUNT-I-CDROM_ISO, MANAGEMENT:VMS_ONLINE_DOCUMENTATION (4 of 4) ,
mounted on  _$1$DKA4: (CDROM)
```

This command mounts four members of an ISO 9660 volume set whose volume set name is VMS_ONLINE_DOCUMENTATION.

9.
```
$ MOUNT/SYSTEM/SUBSYSTEM $8$DKA300: ATLANTIS_WORK1
%MOUNT-I-MOUNTED, ATLANTIS_WORK1 mounted on _$8$DKA300: (ATLANTIS)
$ SHOW DEVICE/FULL $8$DKA300:


Disk $8$DKA300: (ATLANTIS), device type RZ24, is online, mounted,
 file-oriented device, shareable, served to cluster via MSCP Server,
 error logging is enabled.

Error count                    0   Operations completed               385
 Owner process                 ""   Owner UIC                      [SYSTEM]
 Owner process ID        00000000   Dev Prot            S:RWPL,O:RWPL,G:R,W
 Reference count                1   Default buffer size                512
 Total blocks              409792   Sectors per track                   38
 Total cylinders             1348   Tracks per cylinder                  8
 Allocation class               8

 Volume label   "ATLANTIS_WORK1"   Relative volume number                0
 Cluster size                   3   Transaction count                     1
 Free blocks               396798   Maximum files allowed             51224
 Extend quantity                5   Mount count                           1
 Mount status              System   Cache name        "_$8$DKA700:XQPCACHE"
 Extent cache size             64   Maximum blocks in extent cache    39679
 File ID cache size            64   Blocks currently in extent cache      0
 Quota cache size              50   Maximum buffers in FCP cache        295
 Volume owner UIC    [VMS,PLATO]   Vol Prot    S:RWCD,O:RWCD,G:RWCD,W:RWCD


Volume status: ODS-2, subject to mount verification, protected
    subsystems enabled, file high-water marking, write-through caching enabled.
```

The **MOUNT** command mounts a volume labeled ATLANTIS_WORK1, which is available systemwide. Subsystems on the volume are accessible.

10.
```
$ MOUNT DSA0: /SHADOW=($200$DKA200:,$200$DKA300:,$200$DKA400:) X5OZCOPY
%MOUNT-I-MOUNTED, X5OZCOPY mounted on _DSA0:
%MOUNT-I-SHDWMEMSUCC, _$200$DKA200: (VIPER1) is now a valid member of
the shadow set
%MOUNT-I-SHDWMEMSUCC, _$200$DKA300: (VIPER1) is now a valid member of
the shadow set
```

```
%MOUNT-I-SHDWMEMSUCC, _$200$DKA400: (VIPER1) is now a valid member of
the shadow set
$ DISMOUNT DSA0:
$ MOUNT/INCLUDE DSA0: /SHADOW=$200$DKA200: X5OXCOPY
%MOUNT-I-MOUNTED, X5OZCOPY mounted on _DSA0:
%MOUNT-I-SHDWMEMSUCC, _$200$DKA200: (VIPER1) is now a valid member of
the shadow set
%MOUNT-I-AUTOMEMSUCC, _$200$DKA300: (VIPER1) automatically added to the
shadow set
%MOUNT-I-AUTOMEMSUCC, _$200$DKA400: (VIPER1) automatically added to the
shadow set
```

In this example, an existing shadow set is mounted in two ways. The first **MOUNT** command specifies each member of the shadow set with the **/SHADOW** qualifier. Then, after `DSA0:` is dismounted, the second **MOUNT** command uses the **/INCLUDE** qualifier to automatically mount all members of the shadow set.

Lexical Functions

Lexical Functions

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 relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#SYMBOLS_CH].

Table 4, "Summary of Lexical Functions" 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 4. Summary of Lexical Functions

Function	Description
F$CONTEXT	Specifies selection criteria for use with the F$PID function.
F$CSID	Returns an OpenVMS Cluster identification number and updates the context symbol to point to the current position in the system's cluster node list.
F$CUNITS	Converts a number from one specified unit of measure to another.
F$CVSI	Extracts bit fields from character string data and converts the result, as a signed value, to an integer.
F$CVTIME	Retrieves information about an absolute, combination, or delta time string.

Function	Description
F$CVUI	Extracts bit fields from character string data and converts the result, as an unsigned value, to an integer.
F$DELTA_TIME	Returns the time difference between a given start and end time.
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 $FAO system service to convert the specified control string to a formatted ASCII output string.
F$FID_TO_NAME	Translates a file identification to a file specification.
F$FILE_ATTRIBUTES	Returns attribute information fora specified file.
F$GETDVI	Invokes the $GETDVI system service to return a specified item of information for a specified device.
F$GETENV (Alpha only)	Invokes the $GETENV system service to return the value of the specified console environment variable.
F$GETJPI	Invokes the $GETJPI system service to return accounting, status, and identification information for a process.
F$GETQUI	Invokes the $GETQUI system service to return information about queues, batch and print jobs currently in those queues, form definitions, and characteristic definitions kept in the queue database.
F$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$LICENSE	Checks whether the specified license is loaded on the system.
F$LOCATE	Locates a character or character substring within a string and returns its offset within the string.
F$MATCH_WILD	Performs a wildcard matching between a candidate and a pattern string.
F$MESSAGE	Returns the message text associated with a specified system status code value.
F$MODE	Shows the mode in which a process is executing.
F$MULTIPATH	Returns a specified item of information for a specific multipath-capable device.

Function	Description
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$UNIQUE	Generates a string that is suitable to be a file name and is guaranteed to be unique across the cluster.
F$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

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 an OpenVMS Cluster system.

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. Privileges are not required to see processes for the same UIC. To see processes for another UIC in the same UIC group, you need the GROUP privilege, and to see processes systemwide, you need the WORLD privilege.

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$PID. Specifies a keyword that tells F$CONTEXT which selection criterion to use. Use only one selection-item keyword per call to F$CONTEXT.

Note

Do not use the NEQ selection value on a list of items because it causes the condition to always be true.

For example:

```
$ EXAMPLE=F$CONTEXT("PROCESS", CTX,"USERNAME","A*,B*","NEQ")
```

This equation is parsed as "if the user name is not equal to A* *or* the user name is not equal to B*, then return the process of the users that meet the criteria." Because the operand is a logical *or*, the conditions will always be true (any name will be found to be not equal to A* or B*; ALFRED will not be equal to B*; BOB will not be equal to A*).

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. The asterisk (*) and the percent sign (%) wildcard characters are allowed.
AUTHPRI	Integer	GEQ, GTR, LEQ, LSS, EQL, NEQ	On Alpha, valid authorized base priority (0--63).
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 *VSI OpenVMS Guide to System Security* [https://docs.vmssoftware.com/vsi-openvms-guide-to-system-security/].
GRP	Integer	GEQ, GTR, LEQ, LSS, EQL, NEQ	UIC group number.
HW_MODEL	Integer	EQL, NEQ	Valid hardware model number.

Selection Item	Selection Value	Value Qualifiers	Comments
HW_NAME	String	EQL, NEQ	Valid hardware name or a list of keywords. The asterisk (*) and the percent sign (%) wildcard characters 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 *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/].
MASTER_PID	String	EQL, NEQ	PID of master process.
MEM	Integer	GEQ, GTR, LEQ, LSS, EQL, NEQ	UIC member number.
MODE	Keyword	EQL, NEQ	Valid process mode keyword. Valid keywords are OTHER, NETWORK, BATCH, and INTERACTIVE. For more information, see the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/].
NODE_CSID	Integer	EQL, NEQ	Node's cluster ID number.
NODENAME	String	EQL, NEQ	Node name or list of node names. The asterisk (*) and the percent sign (%) 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. The asterisk (*) and the percent sign (%) wildcard characters are allowed.
PRI	Integer	GEQ, GTR, LEQ, LSS, EQL, NEQ	Process priority level number (0--63, on Alpha).
PRIB	Integer	GEQ, GTR, LEQ, LSS, EQL, NEQ	Base process priority level number (0--63, on Alpha).
STATE	Keyword	EQL, NEQ	Valid process state keyword. For more information, see the description of the $GETJPI service in the *VSI OpenVMS System Services Reference Manual: A–GETUAI* [https://docs.vmssoftware.com/vsi-openvms-system-services-reference-manual-a-getuai/#JUN_290].
STS	Keyword	EQL, NEQ	Valid process status keyword. For more information, see the description of the

Selection Item	Selection Value	Value Qualifiers	Comments
			$GETJPI service in the *VSI OpenVMS System Services Reference Manual: A–GETUAI* [https://docs.vmssoftware.com/vsi-openvms-system-services-reference-manual-a-getuai/#JUN_290].
TERMINAL	String	EQL, NEQ	Terminal name or list of names. The asterisk (*) and the percent sign (%) 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. The asterisk (*) and the percent sign (%) wildcard characters 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 the asterisk (*) and the percent sign (%) wildcard characters 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 one of the following process values:

- LSS – less than the value specified in the call to F$PID
- LEQ – less than or equal to the value specified in the call to F$PID
- GTR – greater than the value specified in the call to F$PID
- GEQ – greater than or equal to the value specified in the call to F$PID
- EQL – equal to the value specified in the call to F$PID
- NEQ – not equal to the value specified in the call to F$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 – requires the values to match exactly (that is, values not specified must not be true of the process)
- NEQ – requires that the value must not match

When using multiple selection values with a particular selection qualifier, a match on any one of the selection criteria is considered valid (as if an OR operand was in place); the selection values are not cumulative criteria (as if an AND operand was in place).

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.

Examples

1.
```
$!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.

2.
```
F$CONTEXT("PROCESS",ctx,"prcnam  ","symbiont*,mcote*","eql")

F$CONTEXT("PROCESS",ctx,"prcnam  ","symbiont*,mcote* ","neq")

F$CONTEXT("PROCESS",ctx,"prcnam  ","mcote* ","neq")
F$CONTEXT("PROCESS",ctx,"prcnam  ","symbiont*","neq")
```

This example shows three sets of lexicals showing the difference between the EQL and the NEQ selection values. The first lexical function (with EQL) passes back all processes with symbiont

and `mcote` in the process name. The second and third lexical functions (with `NEQ`) are equivalent in that they both will pass back all processes (processes that do not have `symbiont` in the process name, or processes that do not have `mcote` in the process name).

F$CSID

F$CSID — Returns an identification number from an OpenVMS Cluster system 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 cluster identification number 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 identification number.

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 cluster identification number of the first system in the system's cluster node list. Subsequent calls to the F $CSID function will return the cluster identification number of 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$CSID function call returns the cluster identification number of another node in the cluster. Note that the cluster identification numbers are returned in random order. After the cluster identification number of the last system in the list is returned, the F$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$CUNITS

F$CUNITS — Converts a number from one specified unit of measure to another.

Format

F$CUNITS(*number* [,*from-units*, *to-units*])

Return Value

A number representing the converted value.

Arguments

number

Specifies a 32-bit (or smaller) number to convert.

from-units

Specifies the unit of measure from which to convert. When only first argument is present, the default option for this field is BLOCKS. Supported options for this field are BLOCKS, B, KB, MB, GB, and TB.

to-units

Specifies the unit of measure to which to convert. When only first argument is present, or the second argument is BLOCKS, the default option for this field is BYTES and the result gets rounded off to appropriate *to-unit*. Supported options for this field are BLOCKS, BYTES, B, KB, MB, GB, and TB. Keyword "BYTES" is supported only for BLOCKS to BYTES conversion.

Examples

1. ```
   $ WRITE SYS$OUTPUT F$CUNITS(1024)
   512KB

   $ WRITE SYS$OUTPUT F$CUNITS(1024, "BLOCKS")
   512KB

   $ WRITE SYS$OUTPUT F$CUNITS(1024, "BLOCKS", "BYTES")
   512KB
   ```

   The above examples convert `1024 Blocks` to the equivalent in `Bytes` and auto scale the output. The result is `512 KB`.

2. ```
   $ WRITE SYS$OUTPUT F$CUNITS(1024, "BLOCKS", "B")
   524288B
   ```

   This example converts `1024 Blocks` to non scaled bytes value. The result is `524288 Bytes`.

3. ```
   $ WRITE SYS$OUTPUT F$CUNITS (512,"B", "BLOCKS")
   1BLOCKS
   ```

   This example converts `512 Bytes` to the equivalent in `Blocks`. The result is `1 Blocks.`

4. ```
   $ WRITE SYS$OUTPUT F$CUNITS (10,"KB","B")
   10240B
   ```

   This example converts `10 KB` to the equivalent in `Bytes`. The result is `10240 Bytes`.

5. ```
   $ WRITE SYS$OUTPUT F$CUNITS (1024,"MB","GB")
   1GB
   ```

   This example converts `1024 MB` to the equivalent in `GB`. The result is `1 GB`.

6. ```
   $ WRITE SYS$OUTPUT F$CUNITS(512, "MB", "BLOCKS")
   1048576BLOCKS
   ```

   This example converts `512 MB` to the equivalent in `Blocks`. The result is `1048576 Blocks`.

`CONFLICT` warning message is displayed when keyword `"BYTES"` is used for other than `"BLOCKS"` to `"BYTES"` conversion. For example:

```
$ WRITE SYS$OUTPUT F$CUNITS (512,"BYTES","BLOCKS")
%DCL-W-CONFLICT, illegal combination of command elements - check
 documentation
\BYTES\
$ WRITE SYS$OUTPUT F$CUNITS (10,"KB","BYTES")
%DCL-W-CONFLICT, illegal combination of command elements - check
 documentation
\BYTES\
```

The correct syntax to be used is as follows:

```
$ WRITE SYS$OUTPUT F$CUNITS (512,"B", "BLOCKS")
1BLOCKS
$ WRITE SYS$OUTPUT F$CUNITS (10,"KB","B")
10240B
```

F$CVSI

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 exceeds the number of bits in the string, 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

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 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 specifying time values, see the relevant section in the *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-manual/#DATE_AND_TIME_SECT] or the online help topic Date.

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*. Single-digit days are returned with no leading space or zero.
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.
DELTADCL (V9.2-3 or higher)	The requested information should be returned in the delta time format used by DCL: *dddd-hh:mm:ss.cc*. If you specify DELTADCL as the `output_time_format` argument, then you must 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, DAYOFYEAR, HOUROFYEAR, MINUTEOFYEAR, SECONDOFYEAR.

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, YEAR, DAYOFYEAR, HOUROFYEAR, MINUTEOFYEAR, or SECONDOFYEAR.

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 = "14-DEC-2002 10:56:23.10"
   $ TIME = F$CVTIME(TIME)
   $ SHOW SYMBOL TIME
     TIME = "2002-12-14 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, 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.

3.
```
$ SHOW TIME
  27-MAR-2002 09:50:31
$ WRITE SYS$OUTPUT F$CVTIME(,,"DAYOFYEAR")
86
$ WRITE SYS$OUTPUT F$CVTIME(,,"HOUROFYEAR")
2049
$ WRITE SYS$OUTPUT F$CVTIME(,,"MINUTEOFYEAR")
122991
$ WRITE SYS$OUTPUT F$CVTIME(,,"SECONDOFYEAR")
7379476
```

   In this example, F$CVTIME returns the values for the following keywords: DAYOFYEAR, HOUROFYEAR, MINUTEOFYEAR, and SECONDOFYEAR.

F$CVUI

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. Specifies the character string to be edited.

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$DELTA_TIME

F$DELTA_TIME — Returns the time difference between a given start and end time. The end time must be the same as or later than the start time.

Format

F$DELTA_TIME(*start-time*,*end-time*,*format*)

Return Value

A character string containing the difference between the start and end times. The returned string has the following fixed format:

dddd hh:mm:ss.cc

Arguments

start-time

Absolute time expression of the start time in the following format:

dd-mmm-yyyy hh:mm:ss.cc

end-time

Absolute time expression of the end time in the following format:

dd-mmm-yyyy hh:mm:ss.cc

format

Format for delta time return value. The keywords are as follows:

- ASCTIM: ASCII time format
- DCL: DCL delta time format. This format can be used as an input to other DCL time-related lexicals and commands.

Example

1. ```
   $ START=F$TIME()
   $ END=F$TIME()
   $ SHOW SYMBOL START
   START = "15-JUL-2003 16:26:35.77"
   $ SHOW SYMBOL END
   END = "15-JUL-2003 16:26:41.39"
   $ WRITE SYS$OUTPUT F$DELTA_TIME(START,END)
     0 00:00:05.62
   ```

   This example uses the `F$TIME()` lexical function to define a symbol for the start time and end time. It then uses `F$DELTA_TIME` to display the time difference between the start and end time.

2. ```
   WRITE SYS$OUTPUT F$DELTA_TIME(START,END,"DCL")
   0-00:00:11.91
   $ WRITE SYS$OUTPUT F$DELTA_TIME(START,END,"ASCTIM")
     0 00:00:11.91
   ```

   This example returns the delta between the start and end time in DCL and ASCII formats.

3. ```
   WRITE SYS$OUTPUT F$DELTA_TIME ("BOOT", "LOGIN")
     0 10:24:18.92
   $ WRITE SYS$OUTPUT F$DELTA_TIME ("BOOT", "LOGIN", "DCL")
   0-10:24:18.92
   $ WRITE SYS$OUTPUT F$DELTA_TIME ("BOOT", "LOGIN", "ASCTIM")
     0 10:24:18.92
   ```

   This example returns the delta between the boot and login time in DCL and ASCII formats.

F$DEVICE

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. The asterisk (*) and the percent sign (%) wildcard characters 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 the DVI$_DEVTYPE item in the $GETDVI system service for additional information.

devtype

Specifies the device type for which F$DEVICE is to search. Specify the *devtype* argument as a character string expression that corresponds to a valid type name. See the DVI$_DEVTYPE item in the $GETDVI system service for additional information.

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 asterisk (*) and percent sign (%) wildcard character searches based only on the device name; you must specify a valid character string expression for the device class or 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

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 [CARLEN.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

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 that 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.

Note

When UPCASE is specified with LOWERCASE in an edit-list, UPCASE takes precedence.

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.

2.
```
$ 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.

3.
```
$ UNCOMMENT_LINE = F$EDIT("$ DIR ! THIS IS THE COMMENT", "UNCOMMENT")
$ SHOW SYMBOL UNCOMMENT_LINE
$ UNCOMMENT_LINE = "$ DIR"
```

This example uses the F$EDIT function to remove comments.

F$ELEMENT

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

1.
```
$ 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$ELEMENT call is the string "1". The procedure runs the index, chapter1. In the second iteration, NEXT is incremented to 2 and the result of the F$ELEMENT call is the string "1". The procedure runs the index, chapter2. Processing continues until the result of the F$ELEMENT call is the delimiter specified in the call.

F$ENVIRONMENT

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.

Arguments

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 you to directly invoke images (for example, **RUN** is not allowed). 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.
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

Item	Data Type	Information Returned
		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. If used interactively, the terminal device name is returned.
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.
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 in the *VSI OpenVMS DCL Dictionary: N–Z* [https://docs.vmssoftware.com/vsi-openvms-dcl-dictionary-n-z/#BRASS_88].
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_PREFIX	String	Returns the prefix control string set by means of the **SET PREFIX** command.
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

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 = "PAOLO TESTA"
   $ FIRST = F$EXTRACT(0,5,NAME)
   $ SHOW SYMBOL FIRST
     FIRST = "PAOLO"
   ```

   This portion of a command procedure uses the F$EXTRACT function to extract the first 5 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

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 5, "Summary of FAO Directives" 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 5, "Summary of FAO Directives"* 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 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 5, "Summary of FAO Directives" 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.

Note

Two types of directives that are supported by the $FAO system service are not supported by the DCL F $FAO lexical function. These types are:

- Quadword numeric directives (Q, H, and J), which are not supported in DCL because all DCL numeric values are stored and manipulated as longwords.
- String directives other than the !AS directive, which are not supported in DCL because all DCL strings are stored and manipulated by descriptor.

For further information on the $FAO system service directive, see the relevant section in the *VSI OpenVMS System Services Reference Manual* [https://docs.vmssoftware.com/vsi-openvms-system-services-reference-manual-a-getuai/#JUN_245].

Table 5. 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.

Directive	Argument Type	Description
!^	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.
!%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 <LINE> 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 !-!0UL!1%Cchild!%Echildren!%F
 here",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$FID_TO_NAME

F$FID_TO_NAME — Translates a file identification to a file specification.

Format

```
F$FID_TO_NAME(device-name,file-id)
```

Return Value

A character string containing the file specification.

Arguments

device-name

Specifies the device on which the file resides. You can specify a logical name for the device.

file-id

Specifies the file identification that is to be translated into the correlating file specification.

Example

```
$WRITE SYS$OUTPUT F$FID_TO_NAME("SYS$SYSDEVICE","(2901,33,0)")
DISK$NODE1:[VMS$COMMON.SYSEXE]SHOW.EXE;1
```

This example demonstrates that the file with identifier `"2901,33,0"` on the system disk is file `SHOW.EXE`. Note that you can omit the parentheses around the file identifier, provided it is enclosed by double quotation marks.

F$FILE_ATTRIBUTES

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 6, "F $FILE_ATTRIBUTES Items"*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 anyone of the OpenVMS RMS field names listed in *Table 6, "F $FILE_ATTRIBUTES Items"*.

Description

Use the F$FILE_ATTRIBUTES lexical function in DCL assignment statements and expressions to return file attribute information.*Table 6, "F$FILE_ATTRIBUTES Items"* lists the items you can specify with the F$FILE_ATTRIBUTES function, the information returned, and the data type of this information.

Table 6. F$FILE_ATTRIBUTES Items

Item	Return Type	Information Returned
AI	String	TRUE if after-image (AI) journaling is enabled; FALSE if disabled.
ALQ	Integer	Allocation quantity.
BDT	String	Backup date/time.
BI	String	TRUE if before-image (BI) journaling is enabled; FALSE if disabled.
BKS	Integer	Bucket size.
BLS	Integer	Block size.
CBT	String	TRUE if contiguous-best-try; otherwise FALSE.
CDT	String	Creation date/time.
CTG	String	TRUE if contiguous; otherwise FALSE.
DEQ	Integer	Default extension quantity.
DID	String	Directory ID string.
DIRECTORY	String	Returns TRUE or FALSE. Returns TRUE if it is a directory.
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; otherwise FALSE.
FFB	Integer	First free byte.
FID	String	File ID string.
FILE_LENGTH_HINT	String	Record count and data byte count in the form *(n,m)*, where *n* is the record count and *m* is the data byte count. An invalidated count is specified by a -1 for *n* or *m*.
FSZ	Integer	Fixed control area size.
GBC	Integer	Global buffer count.
GBC32	Integer	Enhanced longword version of global buffer count with a per-file maximum size of about 2.1 billion for indexed files.
GBCFLAGS	String	Per-file management flags for sizing of global buffer cache. Returns PERCENT if global buffer count is expressed as a

Item	Return Type	Information Returned
		percentage, DEFAULT if global buffer size is determined at runtime by an algorithm using two global buffer SYSGEN parameters (GB_CACHEALLMAX and GB_DEFPERCENT); or NONE if no per-file management flags are enabled for the file.
GRP	Integer	Owner group number.
JOURNAL_FILE	String	TRUE if the file is a journal; otherwise FALSE.
KNOWN	String	Known file; returns TRUE or FALSE to indicate whether file is installed with the Install utility (INSTALL). However, returns NOSUCHFILE if a file does not exist (for example, the file has been installed but subsequently deleted).
LOCKED	String	TRUE if a file is deaccessed-locked; otherwise FALSE.
LRL	Integer	Longest record length.
MBM	Integer	Owner member number.
MOVE	String	TRUE if move file operations are enabled; otherwise FALSE.
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.
PRESHELVED	String	TRUE if the file is preshelved; otherwise FALSE.
PRO	String	File protection string.
PVN	Integer	Prolog version number.
RAT	String	Record attributes; returns CR, PRN, FTN, "".
RCK	String	TRUE if read check; otherwise 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.
RVN	Integer	Revision number.
SHELVABLE	String	TRUE if the file is shelvable; otherwise FALSE.
SHELVED	String	TRUE if the file is shelved; otherwise FALSE.
STORED_SEMANTICS	String	ASCII string that represents stored semantics.
UIC	String	Owner user identification code (UIC) string.
VERLIMIT	Integer	Version limit number. The value 32767 indicates that no version limit was set.
WCK	String	TRUE if write check; otherwise FALSE.

File attributes are stored in the file header, which is created from information in OpenVMS RMS control blocks. For more information on OpenVMS RMS control blocks, see the relevant section in the *VSI OpenVMS Record Management Services Reference Manual* [https://docs.vmssoftware.com/vsi-openvms-record-management-services-reference-manual/#PART2_RMS_CONTROL_BLOCKS].

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

F$GETDVI — Returns a specified item of information for a specified device.

Format

```
F$GETDVI(device-name,item[,pathname])
```

Return Value

Either an integer, a longword, or a character string, depending on the item you request. *Table 7, "F $GETDVI Items"* 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 7, "F$GETDVI Items"*.

pathname

Specifies a path name for a multipath-capable device. Specify the path name as a character string expression.

Check the definitions of the item codes in *Table 7, "F$GETDVI Items"* to see if the *pathname* argument is used. In general, item codes that return information that can vary by path do use the *pathname* argument. You can see the paths for a multipath device by using the **SHOW DEVICE /FULL** command, the SYS$DEVICE_PATH_SCAN system service, or the F$MULTIPATH lexical function.

If the *pathname* argument is specified, it is validated against the existing paths for the specified device. If the path does not exist, the NOSUCHPATH error is returned – even if the specified item code does not make use of the *pathname* argument.

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. Unless otherwise stated in the description of the item argument, F$GETDVI returns device information about the local node only.

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 7, "F$GETDVI Items"* lists the items you can specify with the F$GETDVI function, the type of information returned, and the data types of the return values. In addition to the return information listed in *Table 7, "F$GETDVI Items"*, the F$GETDVI lexical function returns any error messages generated by the $GETDVI system service.

For more information on the $GETDVI system service and the items you can specify, see the relevant section in the *VSI OpenVMS System Services Reference Manual: A–GETUAI* [https://docs.vmssoftware.com/vsi-openvms-system-services-reference-manual-a-getuai/#JUN_285].

Table 7. F$GETDVI Items

Item	Return Type	Information Returned[1]
ACCESSTIMES_RECORDED	String	TRUE or FALSE to indicate whether the volume supports the recording of access times.
ACPPID	String	Ancillary control process (ACP) identification.
ACPTYPE	String	ACP type code, as one of the following strings: F11V1, F11V2, F11V3, F11V4, F11V5, F64, HBS, JNL, MTA, NET, REM, UCX, or ILLEGAL. The ACPTYPE item returns ILLEGAL if: ● The device is not mounted or is mounted using the **/FOREIGN** qualifier. ● The ACPTYPE is not currently defined.

Item	Return Type	Information Returned[1]
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.
AVAILABLE_PATH_COUNT	Integer	Number of available, working paths for a multipath-capable device.
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 the Examples section to determine the device class values returned on your system.
DEVDEPEND	Integer	Device-dependent information.
DEVDEPEND2	Integer	Additional device-dependent information.
DEVICE_MAX_IO_SIZE	Integer	The maximum unsegmented transfer size supported by the device's device driver. Although this value is the absolute maximum size supported by the device driver, other software layers (RMS and XFC, for example) might impose lower maximum values, thereby limiting the maximum transfer size.
DEVICE_TYPE_NAME	String	Device type name. Note that if the device is a SCSI tape or disk, the device type name is retrieved directly from the device.
DEVLOCKNAM	String	A unique lock name for the device.
DEVNAM	String	Device name.

Item	Return Type	Information Returned[1]
DEVSTS	Integer	Device-dependent status information.
DEVTYPE	Integer	Device type. See the Examples section to determine the device type values returned on your system.
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.
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.
ERASE_ON_DELETE	String	TRUE or FALSE to indicate whether disk blocks are zeroed upon file deletion on the volume.
ERRCNT	Integer	Error count of the device. If the error count has been reset with the **SET DEVICE /RESET=ERRCNT** command, you can use the **SHOW DEVICE/FULL** command to display the date and time that the error count was reset. If the *pathname* parameter is specified, only the error count for that path is returned. If the *pathname* parameter is omitted, the summation of the error counts for all paths in a multipath device is returned.
ERROR_RESET_TIME	String	Time at which the error count was reset.
EXISTS	String	TRUE or FALSE to indicate whether the device exists on the system.
EXPSIZE	Integer	Current expansion limit on the volume.
FC_HBA_FIRMWARE_REV	String	Firmware revision information of a Fibre Channel host bus adapter. A null string is returned for all other devices.
FC_NODE_NAME	String	The Fibre Channel host bus adapter node name.
FC_PORT_NAME	String	The Fibre Channel host bus adapter port name.
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.

Item	Return Type	Information Returned[1]
GEN	String	TRUE or FALSE to indicate whether the device is a generic device.
HARDLINKS_SUPPORTED	String	TRUE or FALSE to indicate whether hardlinks, rather than aliases, are supported on the volume.
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 OpenVMS Cluster.
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.
LAN_ALL_MULTICAST_MODE	String	TRUE or FALSE to indicate whether the device is enabled to receive all multicast packets rather than only packets addressed to enabled multicast addresses.
LAN_AUTONEG_ENABLED	String	TRUE or FALSE to indicate whether the device is set to autonegotiate the speed and duplex settings.
LAN_DEFAULT_MAC_ADDRESS	String	The default MAC (media access control) address of the device.
LAN_FULL_DUPLEX	String	TRUE or FALSE to indicate whether the device is operating in full-duplex mode.
LAN_JUMBO_FRAMES_ENABLED	String	TRUE or FALSE to indicate whether jumbo frames are enabled on the device.
LAN_LINK_STATE_VALID	String	TRUE or FALSE to indicate whether or not the device driver for the LAN device correctly maintains the link status. The device drivers for the following devices do not maintain the link status: DEMNA, any TURBOchannel adapter, any PCMPIA Ring adapter, Galaxy shared memory, TGEC, DE205, DE422, DE425, DE434, DE435, DE500 (the -XA and -AA variants; only the -BA variant is supported).
LAN_LINK_UP	String	TRUE or FALSE to indicate whether the link is up. This item code is valid only for the template device (that is, unit number 0); this item returns 0 if used with a non-template LAN device. This item is supported only on newer adapters; to determine whether or not a particular device supports LAN_LINK_UP, you must first use F$GETDVI with the item LAN_LINK_STATE_VALID. See the description of LAN_LINK_STATE_VALID

Item	Return Type	Information Returned[1]
		for more information. If LAN_LINK_UP is used on an adapter that does not maintain the link status, the returned status will be SS$_UNSUPPORTED.
LAN_MAC_ADDRESS	String	The current MAC (media access control) address of the device. For more information about the distinction between the default and current MAC addresses, see the relevant section in the *VSI OpenVMS System Services Reference Manual: A–GETUAI* [https://docs.vmssoftware.com/vsi-openvms-system-services-reference-manual-a-getuai/#JUN_285].
LAN_PROMISCUOUS_MODE	String	TRUE or FALSE to indicate whether the device is enabled to receive all packets, rather than only packets addressed to the MAC addresses and to enabled multicast addresses.
LAN_PROTOCOL_NAME	String	The name of the LAN protocol running on the device.
LAN_PROTOCOL_TYPE	String	The type of the LAN protocol running on the device.
LAN_SPEED	Integer	The speed of the LAN device, in units of megabits per second. Valid values are 4, 10, 16, 100, 1000, and 10000.
LOCKID	Integer	Clusterwide lock identification.
LOGVOLNAM	String	Logical volume name.
MAILBOX_BUFFER_QUOTA	Integer	The current mailbox quota as an unsigned integer longword.
MAILBOX_INITIAL_QUOTA	Integer	The initial mailbox quota as an unsigned integer longword.
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	Integer	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.
MOUNT_TIME	String	Time at which the volume was mounted. For volumes mounted in a cluster, only the time of the initial mount is recorded; the time of any subsequent mount is not recorded.

Item	Return Type	Information Returned[1]
MOUNTCNT	Integer	Number of times the volume has been mounted on the local system. The value of MOUNTCNT displayed by the **SHOW DEVICE** command is the total of all mounts of the volume across all members of the cluster.
MOUNTCNT_CLUSTER	Longword	The number of systems in a cluster that have a device mounted. Note that this is not a direct replacement for the MOUNTCNT item, which returns the number of mounters for any device on the local system. The **/SHARE** qualifier to the **MOUNT** command can allow for more than one mounter.
MOUNTVER_ELIGIBLE	String	TRUE or FALSE to indicate whether the volume is eligible to undergo mount verification. A volume mounted with either the **/FOREIGN** or **/NOMOUNT_VERIFICATION** qualifier is not subject to mount verification.
MPDEV_AUTO_PATH_SW_CNT	Integer	Number of times a multipath device has automatically switched paths because of an I/O error or as the result of automatically "failing back" to a local path from a remote path once the local path became available.
MPDEV_CURRENT_PATH	String	Current path name for multipath devices. If the device is not part of a multipath set, this lexical returns the name of the device path if the class driver for this device supports path names. SYS$DKDRIVER, SYS$DUDRIVER, SYS$MKDRIVER, and SYS$GKDRIVER support path names. Returns a null string if the class driver for the device does not support path names.
MPDEV_MAN_PATH_SW_CNT	Integer	Number of times a multipath device has manually switched paths because of a **SET DEVICE /PATH /SWITCH** command or use of the $SET_DEVICE system service.
MT3_DENSITY	String	Current density of the device (tapes only).
MT3_SUPPORTED	String	TRUE or FALSE to indicate whether the device supports densities defined in the MT3DEF (for Alpha tapes only).
MULTIPATH	String	TRUE or FALSE to indicate whether the device is a member of a multipath set.
MVSUPMSG	String	TRUE or FALSE to indicate whether mount verification OPCOM messages are currently being supressed on this device. See the MVSUPMSG_INTVL and MVSUPMSG_NUM system parameters for

Item	Return Type	Information Returned[1]
		more information on the supression of mount verification messages.
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).
NOCACHE_ON_VOLUME	String	TRUE or FALSE to indicate whether the volume is mounted with all caching disabled.
NOHIGHWATER	String	TRUE or FALSE to indicate whether high-water marking is disabled on the volume.
NOSHARE_MOUNTED	String	TRUE or FALSE to indicate whether the volume is mounted with **/NOSHARE**.
ODS2_SUBSET0	String	TRUE or FALSE to indicate whether the volume mounted supports only a subset of the ODS-2 file structure.
ODS5	String	TRUE or FALSE to indicate whether the volume is mounted ODS-5.
ODV	String	TRUE or FALSE to indicate whether the device is capable of providing output.
OPCNT	Integer	Operation count of the device. Note that the operation count may have been reset with the **SET DEVICE/RESET=OPCNT** command. If the *pathname* parameter is specified, only the operation count for that path is returned. If the *pathname* parameter is omitted, the summation of the operation counts for all paths in a multipath device is returned.
OPR	String	TRUE or FALSE to indicate whether the device is an operator.
OWNUIC	String	User identification code (UIC) of the device owner.
PATH_AVAILABLE	String	TRUE or FALSE to indicate whether the specified path is available. This item code is typically used with the *pathname* parameter. If the *pathname* parameter is omitted, information about the current path of the multipath device is returned.
PATH_NOT_RESPONDING	String	TRUE or FALSE to indicate whether the specified path is marked as not responding. This item code is typically used with the *pathname* parameter. If the *pathname* parameter is omitted, information about the current path of the multipath device is returned.
PATH_POLL_ENABLED	String	TRUE or FALSE to indicate whether the specified path is enabled for multipath polling.

Item	Return Type	Information Returned[1]
		This item code is typically used with the *pathname* parameter. If the *pathname* parameter is omitted, information about the current path of the multipath device is returned.
PATH_SWITCH_FROM_TIME	String	Time from which this path was switched, either manually or automatically. This item code is typically used with the *pathname* parameter. If the *pathname* parameter is omitted, information about the current path of the multipath device is returned.
PATH_SWITCH_TO_TIME	String	Time to which this path was switched, either manually or automatically. This item code is typically used with the *pathname* parameter. If the *pathname* parameter is omitted, information about the current path of the multipath device is returned.
PATH_USER_DISABLED	String	TRUE or FALSE to indicate whether the specified path has been disabled using the **SET DEVICE /PATH /NOENABLE** command. This item code is typically used with the *pathname* parameter. If the *pathname* parameter is omitted, information about the current path of the multipath device is returned.
PID	String	Process identification number of the device owner.
PREFERRED_CPU	Integer	Return argument is a 32-bit CPU bit mask with a bit set indicating the preferred CPU. A return argument containing a bit mask of zero indicates that no preferred CPU exists, either because Fast Path is disabled or the device is not a Fast Path capable device. The return argument serves as a CPU bit mask input argument to the $PROCESS_AFFINITY system service. The argument can be used to assign an application process to the optimal preferred CPU.
PREFERRED_CPU_BITMAP	String	A bitmap string of zeros and, at most, a single 1. The 1 in the bitmask represents the number of the CPU to which the device is affinitized. The length of the string determines by how many CPUs are on the system. If there is no 1 in the bitmap string, then either Fast Path is disabled systemwide, or the device is not Fast Path-capable.
PROT_SUBSYSTEM_ENABLED	String	TRUE or FALSE to indicate whether the volume is mounted with protected subsystems enabled.

Item	Return Type	Information Returned[1]
QLEN	Integer	The queue length for the device. This value is the number of I/O requests already in the driver – not the depth of the I/O pending queue.
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.
SCSI_DEVICE_FIRMWARE_REV	String	Firmware revision number of a SCSI disk or SCSI tape. A null string is returned for any other 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_COPIER_NODE	String	The name of the node that is actively performing the copy or merge operation.
SHDW_DEVICE_COUNT	Integer	The total number of devices in the virtual unit, including devices being added as copy targets.
SHDW_GENERATION	String	The current internal revision number for the virtual unit. This value is subject to change.
SHDW_MASTER	String	TRUE or FALSE to indicate whether the device is a virtual unit.

Item	Return Type	Information Returned[1]
SHDW_MASTER_MBR	String	The name of the master member unit that will be used for merge and copy repair operations and for shadow set recovery operations.
SHDW_MASTER_NAME	String	Device name of the virtual unit that represents the shadow set of which the specified device is a member. F$GETDVI returns a null string ("") if the specified device is not a member, or is itself a virtual unit.
SHDW_MBR_COPY_DONE	String	The percent of the copy operation completed on this member unit.
SHDW_MBR_COUNT	String	The number of full source members in the virtual unit. Devices being added as copy targets are not full source members.
SHDW_MBR_MERGE_DONE	String	The percent of the merge operation completed on this member unit.
SHDW_MBR_READ_COST	String	The current value set for the member unit. This value can be modified to use a user-specified value.
SHDW_MEMBER	String	TRUE or FALSE to indicate whether the device is a shadow set member.
SHDW_MERGE_COPYING	String	TRUE or FALSE to indicate whether the device is a merge member of the shadow set.
SHDW_MINIMERGE_ENABLE	String	A value of TRUE indicates that the virtual unit will undergo a mini-merge, not a full merge, if a system in the cluster crashes.
SHDW_NEXT_MBR_NAME	String	Device name of the next member in the shadow set. If you specify a virtual unit, F$GETDVI returns the device name of a member of the shadow set. If you specify the name of a shadow set member unit with the device name and item arguments, F$GETDVI returns the name of the "next" member unit or a null string if there are no more members. To determine all the members of a shadow set, first specify the virtual unit to F$GETDVI; on subsequent calls, specify the member name returned by the previous F$GETDVI call until it has finished, when it returns a null member name. The device name includes the allocation class if the allocation class is not zero; otherwise it includes the device name of the disk controller.
SHDW_READ_SOURCE	String	The name of the member unit that will be used for reads at this time. The unit with the lowest sum total of its queue length and read cost is used. This is a dynamic value.

Item	Return Type	Information Returned[1]
SHDW_SITE	Integer	The site value for the specified device. This value is set by the **SET DEVICE** or **SET SHADOW** command.
SHDW_TIMEOUT	Integer	The user-specified timeout value set for the device. If the user has not set a value by using the SETSHOSHADOW utility, the value of the SYSGEN parameter SHADOW_MBR_TMO is used for member units and the value of MVTIMEOUT is used for virtual units.
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.
TOTAL_PATH_COUNT	Integer	Number of paths for a multipath-capable device.
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).

Item	Return Type	Information Returned[1]
TT_BRDCSTMBX	String	TRUE or FALSE to indicate whether the terminal uses mailbox broadcast messages (terminals only).
TT_CHARSET	Integer	A bitmap indicating the coded character set supported by the terminal.
TT_CRFILL	String	TRUE or FALSE to indicate whether the terminal requires fill after a carriage return (terminals only).
TT_CS_HANGUL	String	TRUE or FALSE to indicate whether the terminal supports the DEC Korean coded character set.
TT_CS_HANYU	String	TRUE or FALSE to indicate whether the terminal supports the DEC Hanyu coded character set.
TT_CS_HANZI	String	TRUE or FALSE to indicate whether the terminal supports the DEC Hanzi coded character set.
TT_CS_KANA	String	TRUE or FALSE to indicate whether the terminal supports the DEC Kana coded character set.
TT_CS_KANJI	String	TRUE or FALSE to indicate whether the terminal supports the DEC Kanji coded character set.
TT_CS_THAI	String	TRUE or FALSE to indicate whether the terminal supports the DEC Thai coded character set.
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).

Item	Return Type	Information Returned[1]
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).
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).

Item	Return Type	Information Returned[1]
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).
TT_SYSPWD	String	TRUE or FALSE to indicate whether the system password is enabled for a particular terminal.

Item	Return Type	Information Returned[1]
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.
VOLCHAR	String	128-bit string (16 bytes) that represents the volume characteristics or capabilities of the mounted device. If a bit is set, the volume is capable of performing the function.
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).
VOLSIZE	Integer	The volume's current logical volume size.
VOLUME_EXTEND_QUANTITY	Integer	Number of blocks to be used as the default extension size for all files on the volume.
VOLUME_MOUNT_GROUP	String	TRUE or FALSE to indicate whether the volume is mounted **/GROUP**.
VOLUME_MOUNT_SYS	String	TRUE or FALSE to indicate whether the volume is mounted **/SYSTEM**.
VOLUME_PENDING_WRITE_ERR	Integer	The number of pending write errors on the volume.
VOLUME_RETAIN_MAX	String	The maximum retention time for the volume, as specified with the DCL command **SET VOLUME/RETENTION**.
VOLUME_RETAIN_MIN	String	The minimum retention time for the volume, as specified with the DCL command **SET VOLUME/RETENTION**.
VOLUME_SPOOLED_DEV_CNT	Integer	The number of devices spooled to the volume.
VOLUME_WINDOW	Integer	The default window size for the volume.
VPROT	String	The volume protection mask.
WCK	String	TRUE or FALSE to indicate whether the device has write checking enabled.
WRITETHRU_CACHE_ENABLED	String	TRUE or FALSE to indicate whether the volume is mounted with write through caching enabled.
WWID	String	Worldwide identifier for a Fibre Channel device.

[1]In addition to the return information listed, the F$GETDVI lexical function returns any error messages generated by the system service $GETDVI.

Examples

1. ```
   $ 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.

2. ```
   $ LIBRARY/EXTRACT=$DCDEF/OUTPUT=$DCDEF.TXT SYS$LIBRARY:STARLET.MLB
   ```

   This example shows how to create a file, $DCDEF.TXT, containing a list of values for device types and device classes from the STARLET library. The device classes begin with 'DC$', and device types begin with 'DT$'.

   Note that most modern SCSI disks and tapes return the generic DEVTYPE code (DT$_GENERIC_DK or DT$_GENERIC_MK), therefore you should use the DEVICE_TYPE_NAME item:

   ```
   $ X=F$GETDVI("XDELTA$DKA0:","DEVICE_TYPE_NAME")
   $ SHOW SYMBOL X  X = "RZ29B"
   ```

3. ```
   $ WRITE SYS$OUTPUT F$GETDVI ( "$1$DGA30", PATH_SWITCH_TO_TIME",
   _$ "PGA0.5000-1FE1-0001=5782" )

   19-MAY-2006 14:47:41.77
   ```

   This example shows the use of the optional path name parameter for F$GETDVI. If a path is not specified, information for the multipath current path is returned. To determine the paths for a multipath device, use the F$MULTIPATH lexical function.

F$GETENV

F$GETENV — Returns the value of the specified console environment variable.

Format

F$GETENV(*itmlst*)

Return Value

Returns the value of the specified console environment variable. You can modify the console environment variables when the system is in console mode. This lexical function allows you to read the contents of these variables when the system is running.

Arguments

itmlst

The defined console environment variable names are:

Auto_action, Boot_dev, Bootdef_dev, Booted_dev, Boot_file, Booted_file, Boot_osflags, Booted_osflags, Boot_reset, Dump_dev, Enable_audit, License, Char_set, Language, Tty_dev
   ```

Description

Returns the value(s) of the specified console environment variable(s).

Example

```
$ dump_device = f$getenv("dump_dev")
$ write sys$output "The dump device for this system is ", dump_device
```

This function writes out the dump device for the system.

F$GETJPI

F$GETJPI — Returns information about the specified process.

Format

`F$GETJPI(`*`pid`*`,`*`item`*`)`

Return Value

Either an integer or a character string, depending on the item you request. *Table 8, "F$GETJPI Items"* 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 an asterisk (*) or percent sign (%) wildcard character 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 8, "F$GETJPI Items"*.

Description

The F$GETJPI lexical function invokes the $GETJPI system service to return information about the specified process.

Note

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.

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 relevant section in the *VSI OpenVMS System*

Services Reference Manual: A–GETUAI [https://docs.vmssoftware.com/vsi-openvms-system-services-reference-manual-a-getuai/#JUN_290].

The F$GETJPI lexical function returns a zero or a null string if the target process is in a suspended or MWAIT (resource wait) state and the item requested is stored in the virtual address space of the process.

You can use the F$GETJPI lexical function to find out whether a process automatically unshelves files.

When you specify the STS2 item code, F$GETJPI returns a 32-bitnumeric value. When you convert this numeric value to binary format, the digit at symbolic bit position PCB$V_NOUNSHELVE shows you the process unshelving default. If the bit is 1, automatic unshelving is turned off; if 0, automatic unshelving is turned on.

*Table 8, "F$GETJPI Items"* lists the items you can specify with the F$GETJPI function, the information returned, and the data type of this information.

Table 8. 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 quota.
CASE_LOOKUP_IMAGE	String	Returns information about the file name lookup case sensitivity of a specified process. This value is set only for the life of the image. Values are BLIND or SENSITIVE. See the relevant section in the Guide to OpenVMS File Applications *[https://docs.vmssoftware.com/guide-to-openvms-file-applications/]* [https://docs.vmssoftware.com/guide-to-openvms-file-applications/#UPPER_LOWER_LETTERS] for additional information.
CASE_LOOKUP_PERM	String	Returns information about the file name lookup case sensitivity of a specified process. This value is

Item	Return Type	Information Returned
		set for the life of the process unless the style is set again. Values are BLIND or SENSITIVE. See the relevant section in the Guide to OpenVMS File Applications *[https://docs.vmssoftware.com/guide-to-openvms-file-applications/]* [https://docs.vmssoftware.com/guide-to-openvms-file-applications/#UPPER_LOWER_LETTERS] for additional information.
CLASSIFICATION	String	Current MAC classification, as a 20-byte padded string, stored in the PSB.
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.
CURRENT_CAP_MASK	Integer	Current capabilities mask for the specified kernel thread. See the **SET PROCESS/CAPABILITIES** command in the *VSI OpenVMS DCL Dictionary: N–Z* [https://docs.vmssoftware.com/vsi-openvms-dcl-dictionary-n-z/#BRASS_78] for additional information.
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.
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.

Item	Return Type	Information Returned
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).
HOME_RAD	Integer	Home resource affinity domain (RAD). RAD is supported on AlphaServer GS series systems and starting from OpenVMS Version 8.4, support is extended to NUMA capable Integrity servers.
IMAGECOUNT	Integer	Number of images that have been run down for the process.
IMAGE_AUTHPRIV	String	Authorized privilege mask of the installed image.
IMAGE_PERMPRIV	String	Permanent (default) privilege mask of the installed image.
IMAGE_WORKPRIV	String	Working (active) privilege mask of the installed image.
IMAGNAME	String	File name of the current image.
IMAGPRIV	String	Privileges with which the current image was installed.
INSTALL_RIGHTS	Integer	Binary content of the install rights list. This item code returns a list of install rights separated by commas.
INSTALL_RIGHTS_SIZE	Integer	Number of bytes needed to store the install rights.
JOBPRCCNT	Integer	Number of subprocesses owned by the job.
JOBTYPE	Integer	Execution mode of the process at the root of the job tree.
KT_LIMIT	Integer	Returns the per-process kernel threads limit for the process.
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 non-interactive 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.
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.

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	Current message mask as established by the **SET MESSAGE** command. If no mask is specified, the default system message mask is described in the $GETMSG system service. For additional information, see the $PUTMSG system service (for message mask bits), and the F$ENVIRONMENT lexical MESSAGE item.
MULTITHREAD	Integer	Current setting for the process (limited by the system setting).
NODENAME	String	The name of the OpenVMS Cluster node on which the process is running.
NODE_CSID	Integer	Cluster ID of the OpenVMS Cluster node on which the process is running.
NODE_VERSION	String	Operating system version number of the OpenVMS Cluster node on which the process is running.
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.
PARSE_STYLE_PERM	String	Values that were set by $SET_PROCESS_PROPERTIESW.
PARSE_STYLE_IMAGE	String	Values that were set by $SET_PROCESS_PROPERTIESW.
PERMANENT_CAP_MASK	Integer	Permanent capabilities mask for the specified kernel thread. See the **SET PROCESS/CAPABILITIES** command in the *VSI OpenVMS DCL Dictionary: N–Z* [https://docs.vmssoftware.com/vsi-openvms-dcl-dictionary-n-z/#BRASS_78] for additional information.
PERSONA_AUTHPRIV	String	Authorized privilege mask of the persona.
PERSONA_ID	Integer	The ID of the persona as a longword integer.
PERSONA_PERMPRIV	String	Permanent (default) privilege mask of the persona.
PERSONA_RIGHTS	Integer	Binary content of the persona rights list. This item code returns a list of persona rights separated by commas.

Item	Return Type	Information Returned
PERSONA_RIGHTS_SIZE	Integer	Number of bytes needed to store the persona rights.
PERSONA_WORKPRIV	String	Privilege mask of the working (active) persona.
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	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	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.
RIGHTS_SIZE	Integer	Number of bytes required to buffer the rights list. The rights list includes both the system rights list and the process rights list.
SCHED_CLASS_NAME	String	Returns the name of the scheduling class if the process is class scheduled, null string if not.
SHRFILLM	Integer	Maximum number of open shared files allowed for the job to which the process belongs.
SEARCH_SYMLINK_PERM	String	Returns one of the following values: • NOWILDCARD • WILDCARD • NOELLIPSIS
SEARCH_SYMLINK_TEMP	String	Returns one of the following values: • NOWILDCARD • WILDCARD • NOELLIPSIS
SITESPEC	Integer	Per-process site-specific longword.

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	First longword of process status flags.
STS2	Integer	Second longword of process status flags.
SUBSYSTEM_RIGHTS	Integer	Binary content of the subsystem rights list. This item code returns a list of subsystem rights separated by commas.
SUBSYSTEM_RIGHTS_SIZE	Integer	Number of bytes needed to store the subsystem rights.
SWPFILLOC	Integer	Location of the swap file.
SYSTEM_RIGHTS	String	Contents of the system rights list for the process. This item code returns a list of identifier names separated by commas.
SYSTEM_RIGHTS_SIZE	Integer	Number of bytes needed to store the system rights.
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.
TOKEN	String	Token size, specified as TRADITIONAL (255 bytes) or EXPANDED (4000 bytes).
TQCNT	Integer	Remaining timer queue entry quota.
TQLM	Integer	Timer queue entry quota.
TT_ACCPORNAM	String	Access port name for the terminal associated with the process.
TT_PHYDEVNAM	String	Physical device name of the terminal associated with the process.
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 volume mount operations that the process has done.
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.

Item	Return Type	Information Returned
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 limit.

If you use the $GETJPI function to request information on the null processor the swapper process, you can specify any of the items in *Table 8, "F$GETJPI Items"* 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	

Examples

1. ```
   $ 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`.

2. ```
   $ X=F$ENVIRONMENT("MESSAGE")
   $ SHOW SYMBOL X
     X = "/FACILITY/SEVERITY/IDENTIFICATION/TEXT"
   $ X=F$GETJPI("0","MSGMASK")
   $ SHOW SYMBOL X
     X = 15   Hex = 0000000F  Octal = 00000000017
   $ SET MESSAGE /NOFACILITY
   $ X=F$ENVIRONMENT("MESSAGE")
   $ SHOW SYMBOL X
     X = "/NOFACILITY/SEVERITY/IDENTIFICATION/TEXT"
   $ X=F$GETJPI("0","MSGMASK")
   $ SHOW SYMBOL X
     X = 7   Hex = 00000007  Octal = 00000000007
   $ SET MESSAGE /FACILITY
   $ X=F$ENVIRONMENT("MESSAGE")
   $ SHOW SYMBOL X
     X = "/FACILITY/SEVERITY/IDENTIFICATION/TEXT"
   $ X=F$GETJPI("0","MSGMASK")
   $ SHOW SYMBOL X
     X = 15   Hex = 0000000F  Octal = 00000000017
   $
   ```

   This example shows the use of the `F$GETJPI MSGMASK` item.

F$GETQUI

F$GETQUI — Returns information about queues, including batch and print jobs currently in the queues, form definitions, and characteristic definitions kept in the queue database. Also returns information about queue managers.

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$GETQUI.
DISPLAY_CHARACTERISTIC	Returns information about a specific characteristic definition or the next characteristic definition in a wildcard operation.
DISPLAY_ENTRY	Returns information about a specific job entry or the next job entry that matches the selection criteria in a wildcard operation. The DISPLAY_ENTRY function code is similar to the DISPLAY_JOB function code int hat both return job information. DISPLAY_JOB, however, requires that a call be made to establish queue context; DISPLAY_ENTRY does not require that queue context be established. Only those entries that match the user-name of the current process will be processed.
DISPLAY_FILE	Returns information about the next file defined for the current job context. Before you make a call to F$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.
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

Function	Description
	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_MANAGER	Returns information about a specific queue manager or the next queue manager in a wildcard operation.
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$GETQUI("DISPLAY_JOB",[*item*],,[*flags*])
DISPLAY_MANAGER	F$GETQUI("DISPLAY_MANAGER",[*item*],*object-id*,[*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 10, "F $GETQUI Items"* 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, QUI$_SEARCH_NUMBER, and QUI$_SEARCH_JOB_NAME input item codes. The *object-id* argument specifies either the name or the number of an object (for example, a specific queue name, job name, or form number) about which F$GETQUI is to return information. The asterisk (*) and the percent sign (%) wildcard characters are allowed for the following functions:

DISPLAY_CHARACTERISTIC
DISPLAY_ENTRY
DISPLAY_FORM
DISPLAY_MANAGER
DISPLAY_QUEUE

By specifying an asterisk (*) or percent sign (%) wildcard character 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 name is used with wildcard characters, 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 $GETQUI system service. Note that keywords in *Table 9, "F$GETQUI Keywords"* can be used only with certain function codes.

Table 9. F$GETQUI Keywords

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_MANAGER 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_ENTRY 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_ENTRY DISPLAY_FILE DISPLAY_JOB DISPLAY_QUEUE	Selects all job file information about the calling batch job (entry), 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_MANAGER 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.

Note

For most operations, read (R) access is required.

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.

Restriction

The GQC that is saved for wildcarded F$GETQUI calls is destroyed if you run any DCL queue-related command, such as **SHOW QUEUE** or **SHOW ENTRY**. To avoid this problem, use the **SPAWN** command to create a new process in which to run the DCL commands.

For more information, see the description of the $GETQUI system service in the *VSI OpenVMS System Services Reference Manual: A–GETUAI* [https://docs.vmssoftware.com/vsi-openvms-system-services-reference-manual-a-getuai/#JUN_304].

The F$GETQUI function returns information on all items that can be specified with the $GETQUI system service. *Table 10, "F$GETQUI Items"* lists the items you can specify with the F$GETQUI function, the information returned, and the data type of this information.

Table 10. F$GETQUI Items

Item	Return Type	Information Returned
ACCOUNT_NAME[1]	String	The account name of the owner of the specified job.

Item	Return Type	Information Returned
AFTER_TIME	String	The system time at or after which the specified job can execute.
ASSIGNED_QUEUE_NAME [1]	String	The name of the execution queue to which the logical queue specified in the call to F$GETQUI is assigned.
AUTOSTART_ON	String	A list of nodes or node device pairs indicating where the queue can start.
BASE_PRIORITY	Integer	The priority at which batch jobs are initiated from a batch execution queue or the priority of a symbiont process that controls output execution queues.
CHARACTERISTICS[1]	String	The characteristics associated with the specified queue or job.
CHARACTERISTIC_NAME	String	The name of the specified characteristic.
CHARACTERISTIC_NUMBER	Integer	The number of the specified characteristic.
CHECKPOINT_DATA[1]	String	The value of the DCL symbol BATCH$RESTART when the specified batch job is restarted.
CLI[1]	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[1]	Integer	The vector of three longwords. The first longword gives the completion status of the specified job. The second and third longwords give additional status about the print 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[1]	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 mixed-architecture OpenVMS Cluster systems. The node name is

Item	Return Type	Information Returned
		specified by the system 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[1]	String	TRUE or FALSE to indicate whether the specified file is checkpointed.
FILE_COPIES[1]	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[1]	Integer	The number of times the specified file has been processed. This item code is applicable only to output execution queues.
FILE_COUNT	Integer	The number of files in a specified job.
FILE_DELETE	String	TRUE or FALSE to indicate whether the specified file is to be deleted after execution of request.
FILE_DEVICE[1]	String	The internal file-device value that uniquely identifies the selected file. This value specifies the following field in the RMS NAM block: NAM$T_DVI (16 bytes)
FILE_DID[1]	String	The internal file-did value that uniquely identifies the selected file. This value specifies the following field in the RMS NAM block: NAM$W_DID (6 bytes)
FILE_DOUBLE_SPACE	String	TRUE or FALSE to indicate whether the symbiont formats the file with double spacing.
FILE_EXECUTING[1]	String	TRUE or FALSE to indicate whether the specified file is being processed.
FILE_FLAG	String	TRUE or FALSE to indicate whether a flag page is to be printed preceding a file.
FILE_FLAGS[1]	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_DELETE

Item	Return Type	Information Returned
		FILE_DOUBLE_SPACE FILE_FLAG FILE_PAGE_HEADER FILE_PAGINATE FILE_PASSALL FILE_TRAILER
FILE_IDENTIFICATION[1]	String	The internal file-identification value that uniquely identifies the selected file. This value specifies the following file-identification field in the RMS NAM block: NAM$W_FID (6 bytes)
FILE_PAGE_HEADER	String	TRUE or FALSE to indicate whether a page header is to be printed on each page of output.
FILE_PAGINATE	String	TRUE or FALSE to indicate whether the symbiont paginates output by inserting a form feed whenever output reaches the bottom margin of the form.
FILE_PASSALL	String	TRUE or FALSE to indicate whether the symbiont prints the file in PASSALL mode.
FILE_SETUP_MODULES[1]	String	The names of the text modules that are to be extracted from the device control library and copied to the printer before the specified file is printed. This item code is meaningful only for output execution queues.
FILE_SPECIFICATION[1]	String	The fully qualified RMS file specification of the file about which F$GETQUI is returning information.
FILE_STATUS[1]	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[1]	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.
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:

Item	Return Type	Information Returned
		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 inlines.
FORM_NAME[1]	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[1]	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 associated with pending jobs in the queue that were skipped during the current call to F$GETQUI. These jobs were not reported because they did not match the selection criterion in effect for the call to F$GETQUI.
INTERVENING_JOBS	Integer	The number of pending jobs in the queue that were skipped during the current call to F$GETQUI.

Item	Return Type	Information Returned
		These jobs were not reported because they did not match the selection criterion in effect for the call to F$GETQUI.
JOB_ABORTING	String	TRUE or FALSE to indicate whether the system is attempting to abort the execution of a job.
JOB_COMPLETION_QUEUE[1]	String	The name of the queue on which the specified job executed.
JOB_COMPLETION_TIME[1]	String	The time at which the execution of the specified job completed.
JOB_COPIES[1]	Integer	The number of times the specified print job is to be repeated.
JOB_COPIES_DONE[1]	Integer	The number of times that the specified print job has been repeated.
JOB_CPU_LIMIT[1]	String	TRUE or FALSE to indicate whether a CPU time limit is specified for the job.
JOB_ERROR_RETENTION[1]	String	TRUE or FALSE to indicate whether the user requested that the specified job be retained in the queue if the job completes unsuccessfully.
JOB_EXECUTING	String	TRUE or FALSE to indicate whether the specified job is executing or printing.
JOB_FILE_BURST[1]	String	TRUE or FALSE to indicate whether a burst page option is explicitly specified for the job.
JOB_FILE_BURST_ONE[1]	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[1]	String	TRUE or FALSE to indicate whether a flag page precedes each file in the job.
JOB_FILE_FLAG_ONE[1]	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[1]	String	TRUE or FALSE to indicate whether a paginate option is explicitly specified for the job.
JOB_FILE_TRAILER[1]	String	TRUE or FALSE to indicate whether a trailer page follows each file in the job.
JOB_FILE_TRAILER_ONE[1]	String	TRUE or FALSE to indicate whether a trailer page follows only the last copy of the last file in the job.
JOB_FLAGS[1]	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_ERROR_RETENTION JOB_FILE_BURST JOB_FILE_BURST_ONE

Item	Return Type	Information Returned
		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_RETENTION_TIME JOB_WSDEFAULT JOB_WSEXTENT JOB_WSQUOTA
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 [1]	String	TRUE or FALSE to indicate whether the log file is deleted after it is printed.
JOB_LOG_NULL[1]	String	TRUE or FALSE to indicate whether a log file is not created.
JOB_LOG_SPOOL[1]	String	TRUE or FALSE to indicate whether the job log file is queued for printing when the job is complete.
JOB_LOWERCASE[1]	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[1]	String	The name of the specified job.

Item	Return Type	Information Returned
JOB_NOTIFY[1]	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[1]	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.
JOB_RETENTION	String	TRUE or FALSE to indicate whether the user requested that the job be retained indefinitely in the queue regardless of the job's completion status.
JOB_RETENTION_TIME[1]	String	Returns the system time until which the user requested the job be retained in the queue. The system time may be expressed in either absolute or delta time format.
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_STALLED	String	TRUE or FALSE to indicate whether the specified job is stalled because the physical device on which the job is printing is stalled.
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.

Item	Return Type	Information Returned
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: JOB_ABORTING JOB_EXECUTING JOB_HOLDING JOB_INACCESSIBLE JOB_REFUSED JOB_REQUEUE JOB_RESTART JOB_RETAINED JOB_STARTING JOB_TIMED_RELEASE JOB_SUSPENDED JOB_PENDING
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[1]	String	TRUE or FALSE to indicate whether a default working set size is specified for the job.
JOB_WSEXTENT[1]	String	TRUE or FALSE to indicate whether a working set extent is specified for the job.
JOB_WSQUOTA[1]	String	TRUE or FALSE to indicate whether a working set quota is specified for the job.
LAST_PAGE[1]	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[1]	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[1]	String	The name of the log file specified for a job. This item code is meaningful only for batch jobs. Use the JOB_LOG_NULL item code to determine whether a log file will be produced.
MANAGER_NAME	String	The queue manager name.

Item	Return Type	Information Returned
MANAGER_NODES	String	The names of the nodes on which the queue manager may run.
MANAGER_STATUS	Integer	The specified queue manager's status flags. To find the settings of each bit in the field, use one of the following items in place of MANAGER_STATUS: MANAGER_FAILOVER MANAGER_RUNNING MANAGER_START_PENDING MANAGER_STARTING MANAGER_STOPPED MANAGER_STOPPING
NOTE[1]	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[1]	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[1]	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[1]	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_JOB_SIZE_MAX PEND_JOB_SIZE_MIN PEND_LOWERCASE_MISMATCH PEND_NO_ACCESS PEND_QUEUE_BUSY PEND_QUEUE_STATE 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.

Item	Return Type	Information Returned
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.
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[1]	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[1]	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_AUTOSTART	String	TRUE or FALSE if the specified queue has been designated as an AUTOSTART queue.
QUEUE_AUTOSTART_INACTIVE	String	TRUE or FALSE if the queue is an autostart queue that will not be automatically started. If TRUE, a **START/QUEUE** or **INIT/QUEUE/START** command must be issued to restart the queue.
QUEUE_AVAILABLE	String	TRUE or FALSE if the queue is processing one or more jobs but is capable of processing one or more additional jobs.
QUEUE_BATCH	String	TRUE or FALSE to indicate whether the queue is a batch queue or a generic batch queue.

Item	Return Type	Information Returned
QUEUE_BUSY	String	TRUE or FALSE if the number of jobs currently executing on the queue equals the job limit for the 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_DIRECTORY	String	The device and directory specification of the queue database directory for the queue manager.
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.
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:

Item	Return Type	Information Returned
		QUEUE_ACL_SPECIFIED QUEUE_AUTOSTART 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_NO_INITIAL_FF QUEUE_PRINTER QUEUE_RECORD_BLOCKING QUEUE_RETAIN_ALL QUEUE_RETAIN_ERROR QUEUE_SWAP QUEUE_TERMINAL QUEUE_WSDEFAULT QUEUE_WSEXTENT QUEUE_WSQUOTA
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 is not processing any jobs and is capable of doing so or whether the generic queue is capable of feeding executor queues.
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.

Item	Return Type	Information Returned
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[1]	String	The name of the specified queue or the name of the queue that contains the specified job.
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:

Item	Return Type	Information Returned
		QUEUE_ALIGNING QUEUE_AUTOSTART QUEUE_AUTOSTART_INACTIVE QUEUE_AVAILABLE QUEUE_BUSY QUEUE_CLOSED QUEUE_IDLE QUEUE_LOWERCASE QUEUE_PAUSED QUEUE_PAUSING QUEUE_REMOTE QUEUE_RESETTING QUEUE_RESUMING QUEUE_SERVER QUEUE_STALLED QUEUE_STARTING QUEUE_STOP_PENDING QUEUE_STOPPED QUEUE_STOPPING QUEUE_UNAVAILABLE
QUEUE_STOP_PENDING	String	TRUE or FALSE if queue will be stopped when jobs currently in progress have completed.
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.
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.
RAD	Integer	Value of the RAD. A value of "-1" indicates no RAD value is attributed to the queue. RAD is supported on AlphaServer GS series systems and starting from OpenVMS Version 8.4, support is extended to NUMA capable Integrity servers.

Item	Return Type	Information Returned
REQUEUE_QUEUE_NAME[1]	String	The name of the queue to which the specified job is reassigned.
RESTART_QUEUE_NAME[1]	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 VMS node on which jobs initiated from the specified queue execute. The node name matches the value of the system parameter SCSNODE for the target node.
SECURITY_INACCESSIBLE	String	TRUE or FALSE to indicate whether the user has read access to the specified queue.
SUBMISSION_TIME[1]	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[1]	String	The user identification code (UIC) of the owner of the specified job.
USERNAME[1]	String	The user name of the owner of the specified job.
WSDEFAULT[1]	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[1]	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[1]	Integer	The working set quota for the specified job or queue. This value is meaningful only for batch jobs and execution and output queues.

[1]Requires Read (R) access if used with one of the function codes:DISPLAY_ENTRY, DISPLAY_JOB, or DISPLAY_FILE.

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")
$ 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 toe stablish 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.

4.
```
$ THIS_NODE = F$EDIT(F$GETSYI("SCSNODE"),"COLLAPSE")
$ TEMP = F$GETQUI("CANCEL_OPERATION")
$ SET NOON
$LOOP:
$ QUEUE = F$GETQUI("DISPLAY_QUEUE","QUEUE_NAME","*","WILDCARD")
$ IF QUEUE .EQS. "" THEN GOTO ENDLOOP
$ IF THIS_NODE .EQS.-
F$GETQUI("DISPLAY_QUEUE","SCSNODE_NAME","*", -
"WILDCARD,FREEZE_CONTEXT")
$ THEN
$ IF .NOT.-
F$GETQUI("DISPLAY_QUEUE","QUEUE_AUTOSTART","*","WILDCARD,FREEZE_CONTEXT")-
THEN START/QUEUE 'QUEUE'
$ ENDIF
$ GOTO LOOP
$ENDLOOP:
$ SET ON
```

This command procedure looks at all queues associated with the local cluster node and starts any queue that is not marked as autostart.

The procedure starts by obtaining the node name of the local system and clearing the F$GETQUI context. In addition, error handling is turned off for the loop so that, if a queue had been started previously, the resulting error from the **START QUEUE** command does not abort the command procedure.

Inside the loop, the F$GETQUI function gets the next queue name in the queue list. If the result is empty, then it has reached the end of the list and it exits the loop.

The next **IF** statement checks to see if the queue runs on the local node. If it does, then the next statement checks to see if the queue is marked as an autostart queue. If that is false, then the queue is started with the start command. The loop is then repeated.

The final command of the procedure restores DCL error handling to the previous setting.

5.
```
$ IF p1.EQS."" THEN INQUIRE p1 "Queue name"
$ TEMP = F$GETQUI("")
$ QLOOP:
$ QNAME = F$GETQUI("DISPLAY_QUEUE","QUEUE_NAME",p1,"WILDCARD")
$ IF QNAME .EQS. "" THEN EXIT
$ WRITE SYS$OUTPUT ""
$ WRITE SYS$OUTPUT "QUEUE: ", QNAME
$ JLOOP:
$ RETAINED = F$GETQUI("DISPLAY_JOB","JOB_RETAINED",,"ALL_JOBS")
$ IF RETAINED .EQS. "" THEN GOTO QLOOP
$ Entry = F$GETQUI("DISPLAY_JOB","ENTRY_NUMBER",, -
"FREEZE_CONTEXT,ALL_JOBS")
$ WRITE SYS$OUTPUT " Entry: ''Entry' Retained: ''RETAINED'"
$ IF RETAINED.EQS."TRUE" THEN DELETE/ENTRY='Entry'
$ GOTO JLOOP
```

This command procedure deletes all retained entries from a nominated queue or queues. Wildcards are allowed.

6.
```
$ WRITE SYS$OUTPUT F$GETQUI("DISPLAY_QUEUE","RAD","BATCHQ1")
 -1
```

This example returns the value of the RAD. A value of "-1" indicates no RAD value is attributed to the queue.

F$GETSYI

F$GETSYI — Returns status and identification information about the local system (or about a node in the local mixed-architecture OpenVMS Cluster system, if your system is part of an OpenVMS Cluster).

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 OpenVMS Cluster, if your system is part of an OpenVMS Cluster). Specify the item as a character string expression.

You can also specify any system parameter as the item to return the current value of this parameter for the node. For a list and description of all system parameters, refer to the relevant section in the *VSI OpenVMS System Manager's Manual, Volume 2: Tuning, Monitoring, and Complex Systems* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-2-tuning-monitoring-and-complex-systems/#_6017PARAMS].

node-name

Specifies the node in your OpenVMS Cluster system for which information is to be returned. Specify the node as a character string expression. You cannot use the asterisk (*) and the percent sign (%) wildcard characters to specify the node-name argument.

cluster-id

Specifies the cluster node identification number for which the information is to be returned.

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 $GETSYI system service to return status and identification information about the local system (or about a node in the local OpenVMS Cluster, if your system is part of a cluster). The F$GETSYI function returns information on the items that can be specified with the $GETSYI system service. For more information about the $GETSYI system service, see the relevant section in the *VSI OpenVMS System Services Reference Manual: A–GETUAI* [https://docs.vmssoftware.com/vsi-openvms-system-services-reference-manual-a-getuai/#JUN_312].

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 11, "*F$GETSYI *Items"* lists the items you can specify with the F$GETSYI lexical function.

Table 11. F$GETSYI Items

Item	Return Type	Information Returned
ACTIVE_CPU_BITMAP	String	A bitmap with a bit indicating a member of the instance's active set – those currently participating in the OpenVMS SMP scheduling activities. The size of the returned bitmap is determined by the number of supported CPUs on the system. To compute the number of bytes needed for the bitmap, use the F$GETSYI function with the MAX_CPUS item to find the minimum number of bits needed, round this number up to a multiple of 64, and divide the result by 8.
ACTIVE_CPU_MASK	Integer	Note that this item is becoming obsolete; VSI recommends that you *not* use it because it represents only up to 64 CPUs. The service continues to return the correct data for systems with up to 64 CPUs but fails for systems with more than 64 CPUs. For greater flexibility, use item ACTIVE_CPU_BITMAP instead. On Alpha and Integrity server systems, returns a value that represents a CPU-indexed bitvector. When a particular bit position is set, the processor with that CPU ID value is a member of the instance's active set

Item	Return Type	Information Returned
		– those currently participating in the OpenVMS SMP scheduling activities.
ACTIVECPU_CNT	Integer	Count of CPUs actively participating in the current boot of a symmetric multiprocessing (SMP) system.
ARCHFLAG	Integer	Architecture flags for the system.
ARCH_NAME	String	Name of CPU architecture on which the process is executing: "x86_64" for OpenVMS x86-64, "IA64" for OpenVMS Integrity servers, "Alpha" for OpenVMS Alpha, and "VAX" for VAX.
ARCH_TYPE	Integer	Type of CPU architecture on which the process is executing: 4 for OpenVMS x86-64 servers, 3 for Integrity, 2 for Alpha, and 1 for VAX.
AVAIL_CPU_BITMAP	String	A bitmap with a bit indicating a member of the instance's configure set – those owned by the partition and controlled by the issuing instance. The size of the returned bitmap is determined by the number of supported CPUs on the system. To compute the number of bytes needed for the bitmap, use the F$GETSYI function with the MAX_CPUS item to find the minimum number of bits needed, round this number up to a multiple of 64, and divide the result by 8.
AVAIL_CPU_MASK	Integer	Note that this item code is becoming obsolete; VSI recommends that you *not* use it because it represents only up to 64 CPUs. The service continues to return the correct data for systems with up to 64 CPUs but fails for systems with more than 64 CPUs. For greater flexibility, use item AVAIL_CPU_BITMAP instead. On Alpha and Integrity server systems, returns a value that represents a CPU-indexed bitvector. When a particular bit position is set, the processor with that CPU ID value is a member of the instance's configure set – those owned by the partition and controlled by the issuing instance.
AVAILCPU_CNT	Integer	Number of CPUs available in the current boot of the symmetric multiprocessing (SMP) system.
BOOT_DEVICE	String	Name of the device from which the system was booted. For a system with a shadowed system disk, BOOT_DEVICE returns the name of the member device from which the shadow set was formed.
BOOTTIME	String	The time the system was booted.
CHARACTER_EMULATED	String	TRUE if the character string instructions are emulated on the CPU and FALSE if they are not.
CLUSTER_EVOTES	Integer	Number of votes expected to be found in the OpenVMS Cluster system. The cluster determines this value by selecting the highest number from

Item	Return Type	Information Returned
		all of the following: each node's system parameter EXPECTED_VOTES, the sum of the votes currently in the cluster, and the previous value for the number of expected votes.
CLUSTER_FSYSID	String	System identification of the founding node, which is the first node in the OpenVMS Cluster system to boot. The cluster management software assigns this system identification to the node. You can obtain this information by using the DCL command **SHOW CLUSTER**.
CLUSTER_FTIME	String	The time when the founding node is booted. The founding node is the first node in the OpenVMS Cluster system to boot.
CLUSTER_MEMBER	String	TRUE if the node is a member of the local cluster and FALSE if it is not.
CLUSTER_NODES	Integer	Number of nodes currently in the OpenVMS Cluster system.
CLUSTER_QUORUM	Integer	The number that is the total of the quorum values held by all nodes in the OpenVMS Cluster system. Each node's quorum value is derived from its system parameter EXPECTED_VOTES.
CLUSTER_VOTES	Integer	Total number of votes held by all nodes in the OpenVMS Cluster system. The number of votes held by any one node is determined by that node's system parameter VOTES.
[1]CONSOLE_VERSION	String	On Alpha systems, returns the console firmware version.
COMMUNITY_ID	Integer	Hardware community ID for the issuing instance within the hard partition. Supported only on AlphaServer systems that support partitioning.
CONTIG_GBLPAGES	Integer	Total number of free, contiguous global CPU-specific pages. This number is the largest size global section that can be created.
CPU	Integer	CPU processor type of the node. For information about extended processor types, see the description for the XCPU item.
CPU_AUTOSTART	String	List of zeroes and ones, separated by commas and indexed by CPU ID. Any entry with a value of one indicates that specific CPU will be brought into the OpenVMS active set if it transitions into the current instance from outside, or is powered up while already owned.

<table>
<tr><th>Item</th><th>Return Type</th><th>Information Returned</th></tr>
<tr><td>CPU_FAILOVER</td><td>String</td><td>List of numeric partition IDs, separated by commas and indexed by CPU ID, that define the destination of the processor if the current instance should crash.

Supported only on AlphaServer systems that support partitioning.</td></tr>
<tr><td>CPUCAP_MASK</td><td>String</td><td>Array of quadword user capability masks for all CPUs in the system. This array is indexed by CPU ID and contains as many elements as the amount of space specified by the buffer length field in the item descriptor.

To minimize wasted space, a prior call to F$GETSYI with MAX_CPUS will provide the number of CPUs that need to be retrieved. Multiplying that value by 8 bytes for each quadword provides the value to be written in the buffer length field of the item descriptor.</td></tr>
<tr><td>CPUCONF</td><td>Integer</td><td>Note that this item is becoming obsolete; VSI recommends that you not use it because it represents only up to 64 CPUs. The service continues to return the correct data for systems with up to 64 CPUs but fails for systems with more than 64 CPUs. For greater flexibility, use the AVAIL_CPU_BITMAP item instead.

On Alpha and Integrity server systems, returns a value that represents a CPU-indexed bitvector. When a particular bit position is set, the processor with that CPU ID value is a member of the instance's configure set – those owned by the partition and controlled by the issuing instance.</td></tr>
<tr><td>CPUTYPE</td><td>Integer</td><td>Processor type, as stored in the hardware restart parameter block (HWRPB).

For example, the value of 2 represents a DECchip 21064 processor. Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

The following table shows the processor codes and processors:
<table>
<tr><th>Processor Code</th><th>Processor</th></tr>
<tr><td>2</td><td>21064</td></tr>
<tr><td>4</td><td>21066, 21068, 21066A, 21068A</td></tr>
<tr><td>5</td><td>21164</td></tr>
<tr><td>6</td><td>21064A</td></tr>
</table>
</td></tr>
</table>

Item	Return Type	Information Returned	
		7	21164A
		8	21264
		11	21264A
		12	21264C
		13	21264B
		14	21264D
		15	21364
		16	21364
		. . .	
		31	Itanium 2
		32	Itanium 3
CWLOGICALS	String	TRUE if the clusterwide logical name database has been initialized on the CPU and FALSE if it has not been initialized.	
DAY_OVERRIDE	String	TRUE if the **SET DAY** command has been used to override the default primary and secondary day types in the user authorization file that are used to control user logins. FALSE if no override is currently in effect, and the contents of user authorization file records for each user are being honored.	
DAY_SECONDARY	String	TRUE if any override with the **SET DAY** command has been used to specify that the current day is to be considered a Secondary day for user login purposes. FALSE if any override with the **SET DAY** command has been used to specify that the current day is to be considered a Primary day for user login purposes. If F$GETSYI returns FALSE for DAY_OVERRIDE, the number returned for DAY_SECONDARY is meaningless.	
DECIMAL_EMULATED	String	TRUE if the decimal string instructions are emulated on the CPU and FALSE if they are not.	
DECNET_FULLNAME	String	The node name of a DECnet Phase IV system or the node full name of a DECnet-Plus system.	
DECNET_VERSION	String	The information on the particular version and ECO level of the DECnet package installed on the local system. This item returns a string containing a hexadecimal number, using the following format: ● Byte 0 = Customer ECO ● Byte 1 = DECnet ECO	

Item	Return Type	Information Returned
		• Byte 2 = DECnet phase (4 for Phase IV, 5 for DECnet-Plus for OpenVMS) • Byte 3 = Reserved To distinguish Phase IV from DECnet-Plus for OpenVMS, use the byte containing the DECnet version (byte 2).
D_FLOAT_EMULATED	String	TRUE if the D_floating instructions are emulated on the CPU and FALSE if they are not.
DEF_PRIO_MAX	Integer	The maximum priority for the default scheduling policy.
DEF_PRIO_MIN	Integer	The minimum priority for the default scheduling policy.
ERLBUFFERPAGES	Integer	Number of pagelets on Alpha and Integrity servers used for each S0 error log buffer.
ERLBUFFERPAG_S2	Integer	Number of system pagelets (on Alpha and Integrity servers used for each S2 error log buffer).
ERRORLOGBUFFERS	Integer	Number of S0 error log buffers.
ERRORLOGBUFF_S2	Integer	Number of S2 error log buffers.
F_FLOAT_EMULATED	String	TRUE if the F_floating instructions are emulated on the CPU and FALSE if they are not.
FREE_GBLPAGES	Integer	The current count of free global pages.
FREE_GBLSECTS	Integer	The current count of free global section table entries.
FREE_PAGES	Integer	Total number of free pages.
G_FLOAT_EMULATED	String	TRUE if the G_floating instructions are emulated on the CPU and FALSE if they are not.
[1]GALAXY_ID	Integer	The 128-bit Galaxy ID. Supported only on AlphaServer GS series systems.
[1]GALAXY_MEMBER	Integer	1 if you are member of a Galaxy sharing community, 0 if not. Supported only on AlphaServer GS series systems.
[1]GALAXY_PLATFORM	Integer	1 if you are running on a Galaxy platform, 0 if not. Supported only on AlphaServer GS series systems.
[1]GALAXY_SHMEMSIZE	Integer	Number of shared memory pages. If the current instance is not a member of a Galaxy, no shared memory is reported. Supported only on AlphaServer GS series systems.
GH_RSRVPGCNT	Integer	Number of pages covered by granularity hints to reserve for use by the Install utility after system startup has completed.

Item	Return Type	Information Returned
[1]GLX_FORMATION	String	A time-stamp string when the Galaxy configuration, of which this instance is a member, was created. Supported only on AlphaServer GS series systems.
[1]GLX_MAX_MEMBERS	Integer	The maximum count of instances that may join the current Galaxy configuration. Supported only on AlphaServer GS series systems.
[1]GLX_MBR_MEMBER	Integer	A 64-byte integer. Each 8 bytes represents a Galaxy member number, listed from 7 to 0. The value is 1 if the instance is currently a member, 0 if not a member. Supported only on AlphaServer GS series systems.
[1]GLX_MBR_NAME	String	A string indicating the names which are known in the Galaxy membership. Supported only on AlphaServer GS series systems.
[1]GLX_TERMINATION	String	A time-stamp string when the Galaxy configuration, of which this instance last was a member, was terminated. Supported only on AlphaServer GS series systems.
H_FLOAT_EMULATED	String	TRUE if the H_floating instructions are emulated on the CPU and FALSE if they are not.
HP_ACTIVE_CPU_CNT	Integer	Count of CPUs in the hard partition that are not currently in firmware console mode. For OpenVMS, this implies that the CPU is in, or in the process of joining, the active set in one of the instances in the hard partition. Supported only on AlphaServer systems that support partitioning.
HP_ACTIVE_SP_CNT	Integer	Count of active operating system instances currently executing within the hard partition. Supported only on AlphaServer systems that support partitioning.
HP_CONFIG_SBB_CNT	Integer	Count of the existing system building blocks within the current hard partition. Supported only on AlphaServer systems that support partitioning.
HP_CONFIG_SP_CNT	Integer	The maximum count of soft partitions within the current hard partition. This count does not imply that an operating system instance is currently running within any given soft partition. Supported only on AlphaServer systems that support partitioning.

Item	Return Type	Information Returned
HW_MODEL	Integer	A small integer that can be used to identify the model type of the node. An integer greater than 1023 indicates an Alpha and Integrity servers node. An integer less than or equal to 1023 indicates a VAX node.
HW_NAME	String	The model name string of the node. The model name is a character string that describes the model of the node. The model name usually corresponds to the nameplate that appears on the outside of the CPU cabinet.
IO_PRCPU_BITMAP	String	A bitmap with a bit indicating a preferred CPU – one available for Fast Path operations. The size of the returned bitmap is determined by the number of supported CPUs on the system. To compute the number of bytes needed for the bitmap, use the F$GETSYI function with the MAX_CPUS item to find the minimum number of bits needed, round this number up to a multiple of 64, and divide the result by 8.
ITB_ENTRIES	Integer	Number of instruction stream translation buffer entries that support granularity hints to be allocated for resident code.
MAX_CPUS	Integer	The maximum number of CPUs that could be recognized by this instance.
MAX_PFN	Integer	The highest numbered PFN in use by the operating system. The highest numbered PFN used by OpenVMS is influenced by the PHYSICAL_MEMORY system parameter.
MEMSIZE	Integer	Number of pages of memory in the system configuration.
MODIFIED_PAGES	Integer	Total number of modified pages.
MULTITHREAD	Integer	Value of the MULTITHREAD system parameter.
NODENAME	String	The Node name. It does *not* include the following double colon (::).
NODE_AREA	Integer	The DECnet area for the target node.
NODE_CSID	String	The the OpenVMS Cluster system ID (CSID) of the node, as a string containing a hexadecimal number. The CSID is a form of system identification.
NODE_EVOTES	Integer	Number of votes the node expects to find in the OpenVMS Cluster system. This number is determined by the system parameter EXPECTED_VOTES.
NODE_HWVERS	String	Hardware version of the node.

Item	Return Type	Information Returned
NODE_NUMBER	Integer	The DECnet number of the node.
NODE_QUORUM	Integer	Value of the quorum held by the node. This number is derived from the node's system parameter EXPECTED_VOTES.
NODE_SWINCARN	String	Software incarnation number for the node. This number is returned as a string containing a hexadecimal number.
NODE_SWTYPE	String	Type of operating system software used by the node. The operating system software type indicates whether the node is an Alpha or Integrity servers system, or an HSC storage controller.
NODE_SWVERS	String	Software version of the node.
NODE_SYSTEMID	String	System identification number of the node. This number is returned as a string containing a hexadecimal number.
NODE_VOTES	Integer	Number of votes the node expects to find in the OpenVMS Cluster system. This number is determined by the system parameter EXPECTED_VOTES.
[1]NPAGED_FREE	Integer	Number of free bytes in nonpaged pool.
[1]NPAGED_INUSE	Integer	Total number of bytes currently being used in nonpaged pool.
[1]NPAGED_LARGEST	Integer	Size of the largest contiguous area of free memory in nonpaged pool.
[1]NPAGED_TOTAL	Integer	Total size (in bytes) of nonpaged pool.
[1]PAGED_FREE	Integer	Number of free bytes in paged pool.
[1]PAGED_INUSE	Integer	Total number of bytes currently being used in paged pool.
[1]PAGED_LARGEST	Integer	Size of the largest contiguous area of free memory in paged pool.
[1]PAGED_TOTAL	Integer	Total size (in bytes) of paged pool.
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.
PAGE_SIZE	Integer	Number of CPU-specific bytes per page in the system. On Alpha and Integrity server systems, CPU page size varies from system to system.
[1]PALCODE_VERSION	String	Version of the PALCODE (privileged architectural library) on your Alpha system.
PARTITION_ID	Integer	The soft partition ID. Supported only on AlphaServer systems that support partitioning.

Item	Return Type	Information Returned
[1]PFN_MEMORY_MAP	String	A map describing the system's use of physical memory.
PFN_MEMORY_MAP_64	String	A map describing the system's use of physical memory on 64-bit systems.
PHYSICALPAGES	Integer	Total number of PFNs that exist between the first PFN (typically PFN 0) and the highest numbered PFN.
PMD_COUNT	Integer	Total number of physical memory descriptors defined by the system. The return value of this parameter can be used to determine the buffer size to use when specifying the PFN_MEMORY_MAP item.
POTENTIAL_CPU_BITMAP	String	A bitmap with a bit indicating a member of the instance's potential set. A CPU in the potential set implies that it could actively join the OpenVMS active set for this instance if it is ever owned by it. To meet this rule the CPU's characteristics must match hardware and software compatibility rules defined particularly for that instance. The size of the returned bitmap is determined by the number of supported CPUs on the system. To compute the number of bytes needed for the bitmap, use the F$GETSYI function with the MAX_CPUS item to find the minimum number of bits needed, round this number up to a multiple of 64, and divide the result by 8.
POTENTIAL_CPU_MASK	Integer	Note that this item is becoming obsolete; VSI recommends that you *not* use it because it represents only up to 64 CPUs. The service continues to return the correct data for systems with up to 64 CPUs but fails for systems with more than 64 CPUs. For greater flexibility, use item POTENTIAL_CPU_BITMAP instead. On Alpha and Integrity server systems, returns a value that represents a CPU-indexed bit vector. When a particular bit position is set, the processor with that CPU ID value is a member of the instance's potential set. A CPU in the potential set implies that it could actively join the OpenVMS active set for this instance if it is ever owned by it. To meet this rule, the CPU's characteristics must match hardware and software compatibility rules defined particularly for that instance.
POTENTIALCPU_CNT	Integer	Count of CPUs in the hard partition that are members of the potential set for this instance. A CPU in the potential set implies that it could actively joint he OpenVMS active set for this instance if it is ever owned by it. To meet this rule the CPU's

Item	Return Type	Information Returned
		characteristics must match hardware and software compatibility rules defined particularly for that instance.
POWERED_CPU_BITMAP	String	A bitmap with a bit indicating a member of the instance's powered set – those CPUs physically existing within the hard partition and powered up for operation. The size of the returned bitmap is determined by the number of supported CPUs on the system. To compute the number of bytes needed for the bitmap, use the F$GETSYI function with an the MAX_CPUS item to find the minimum number of bits needed, round this number up to a multiple of 64, and divide the result by 8.
POWERED_CPU_MASK	Integer	Note that this item is becoming obsolete; VSI recommends that you *not* use it because it represents only up to 64 CPUs. The service continues to return the correct data for systems with up to 64 CPUs but fails for systems with more than 64 CPUs. For greater flexibility, use item POWERED_CPU_BITMAP instead. On Alpha and Integrity server systems, returns a value that represents a CPU-indexed bitvector. When a particular bit position is set, the processor with that CPU ID value is a member of the instance's powered set – those CPUs physically existing within the hard partition and powered up for operation.
POWEREDCPU_CNT	Integer	Count of CPUs in the hard partition that are physically powered up.
PRESENT_CPU_BITMAP	String	A bitmap with a bit indicating a member of the instance's present set – those CPUs physically existing within the hard partition. Being in the present set does not imply that it is part of the powered set. The size of the returned bitmap is determined by the number of supported CPUs on the system. To compute the number of bytes needed for the bitmap, use the F$GETSYI system service with the MAX_CPUS item to find the minimum number of bits needed, round this number up to a multiple of 64, and divide the result by 8.
PRESENT_CPU_MASK	Integer	Note that this item is becoming obsolete, and VSI recommends that you *not* use it because it represents only up to 64 CPUs. The service continues to return the correct data for systems with up to 64 CPUs but fails for systems with more than 64 CPUs. For greater flexibility, use the PRESENT_CPU_BITMAP item instead.

Item	Return Type	Information Returned
		On Alpha and Integrity server systems, returns a value that represents a CPU-indexed bitvector. When a particular bit position is set, the processor with that CPU ID value is a member of the instance's present set – those CPUs physically existing within the hard partition. Being in the present set does not imply that it is part of the powered set.
PRESENTCPU_CNT	Integer	Count of CPUs in the hard partition that physically reside in a hardware slot.
PRIMARY_CPUID	Integer	The CPU ID of the primary processor for this OpenVMS instance.
PROCESS_SPACE_LIMIT	String	The 64-bit virtual address succeeding the last available process private address. The value returned is the upper bound on the process private address space. The value returned is the same for every process on the system.
PSXFIFO_PRIO_MAX	Integer	The maximum priority for the POSIX FIFO scheduling policy.
PSXFIFO_PRIO_MIN	Integer	The minimum priority for the POSIX FIFO scheduling policy.
PSXRR_PRIO_MAX	Integer	The maximum priority for the POSIX round-robin scheduling policy.
PSXRR_PRIO_MIN	Integer	The minimum priority for the POSIX round-robin scheduling policy.
PT_BASE	String	The 64-bit virtual address of the base of the page tables. The value returned is the same for every process on the system.
PTES_PER_PAGE	Integer	The maximum number of CPU-specific pages that can be mapped by one page table page.
QUANTUM	Integer	The maximum amount of processor time a process can receive while other processes are waiting.
RAD_CPUS	String	A longword array of RAD/CPU pairs that can potentially be in this operating system instance. If there is no RAD support, all potential CPUs are in RAD 0. The array is terminated with a -1,-1 pair. RAD is supported on AlphaServer GS series systems and starting from OpenVMS Version 8.4, support is extended to NUMA capable Integrity servers.
RAD_MEMSIZE	String	A longword array of RAD/page count pairs. The number of pages of private memory is returned. If there is no RAD support, all memory is reported in RAD 0. The array is terminated with a -1,-1 pair. RAD is supported on AlphaServer GS series systems and starting from OpenVMS Version 8.4, support is extended to NUMA capable Integrity servers.

Item	Return Type	Information Returned
RAD_MAX_RADS	Integer	The maximum number of RADS possible on this platform. If there is no RAD support, 1 is returned. RAD is supported on AlphaServer GS series systems and starting from OpenVMS Version 8.4, support is extended to NUMA capable Integrity servers.
RAD_SHMEMSIZE	String	A longword array of RAD/page count pairs. The number of pages of shared memory is returned. If there is no RAD support, all shared memory is reported in RAD 0. If the current instance is not a member of a Galaxy, no shared memory is reported. The array is terminated with a -1,-1 pair. RAD is supported on AlphaServer GS series systems and starting from OpenVMS Version 8.4, support is extended to NUMA capable Integrity servers.
REAL_CPUTYPE	Integer	The actual CPU type of the primary CPU of the system.
SCSNODE	String	The Galaxy instance name. Supported only on AlphaServer systems that support partitioning.
SCS_EXISTS	String	TRUE if the System Communication Subsystem (SCS) is currently loaded on the node and FALSE if the SCS is not currently loaded.
SERIAL_NUMBER	String	System serial number from the Hardware Restart Parameter Block (HWRPB).
SHARED_VA_PTES	String	The 64-bit virtual address of the PTE that marks the boundary between process-private PTEs and system-shared PTEs. The value returned is the same for every process on the system.
SID	Integer	System identification register. On Alpha and Integrity server systems, SID returns a value where all fields are zero except the CPU-type field, which always contains the value of 256.
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 returns a list of identifier names separated by commas (,).
[2]SYSTEM_UUID	Integer	The 128-bit Universal Unique Identifier (UUID) for the system.
SYSTYPE	Integer	The name of the family or system hardware platform. For example, the integer 2 represents a DEC 4000

Item	Return Type	Information Returned
		processor, the integer 3 represents a DEC 7000 or DEC 10000 processor, and the integer 4 represents a DEC 3000 processor.
TOTAL_PAGES	Integer	Total number of physical memory pages.
USED_GBLPAGCNT	Integer	Number of pages currently in use in the global page table.
USED_GBLPAGMAX	Integer	The maximum number of pages ever in use in the global page table.
USED_PAGES	Integer	Total number of used pages.
VERSION	String	Software version number of the OpenVMS operating system running on the node.
VECTOR_EMULATOR	Integer	A byte, the low-order bit of which, when set, indicates the presence of the Vector Instruction Emulator facility (VVIEF) in the system.
[3]VIRTUAL_MACHINE	String	TRUE if OpenVMS is running as a virtual machine guest, FALSE if it is not.
VP_MASK	Integer	A longword mask, the bits of which, when set, indicate which processors in the system have vector coprocessors.
VP_NUMBER	Integer	Number of vector processors in the system.
XCPU	Integer	The extended CPU processor type of the node. You should obtain the general processor type value first by using the CPU item. For some of the general processor types, extended processor type information is provided by the XCPU item. For other general processor types, the value returned by the XCPU item is currently undefined.
XSID	Integer	Processor-specific information. For the MicroVAX II system, this information is the contents of the system type register of the node. The system type register contains the full extended information used in determining the extended system type codes. For other processors, the data returned by XSID is currently undefined.
<SYSTEM_PARAMETER>	Integer or string, depending on the system parameter you request	The current value of the system parameter that you specify as the item for the F$GETSYI lexical function. For a list and description of all system parameters, refer to the relevant section in the *VSI OpenVMS System Manager's Manual, Volume 2: Tuning, Monitoring, and Complex Systems* [https://docs.vmssoftware.com/vsi-openvms-system-manager-

Item	Return Type	Information Returned
		s-manual-volume-2-tuning-monitoring-and-complex-systems/#_6017PARAMS].

[1]Alpha only
[2]Integrity only
[3]Integrity and x86-64 only

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 cluster. The return value TRUE indicates that the remote node LONDON is a member of the cluster.

3.
```
$ LIM = F$GETSYI("IJOBLIM")
$ SHOW SYMBOL LIM
  LIM = 16   Hex = 00000010  Octal = 00000000020
```

   This example uses the system parameter IJOBLIM as an argument for the F$GETSYI function. This argument returns the batch job limit for the current system.

4.
```
$ DECNETVERS = F$GETSYI("DECNET_VERSION")
$ SHOW SYMBOL DECNETVERS
  DECNETVERS = "00050D01"
$ DECNETPHASE = F$INTEGER(F$EXTRACT(2,2,DECNETVERS))
$ SHOW SYMBOL DECNETPHASE
  DECNETPHASE = 5   Hex = 00000005  Octal = 00000000005
```

   This example shows how to use F$GETSYI to return the DECnet version, using the DECNET_VERSION item.

5.
```
$ RADCPU = F$GETSYI("RAD_CPUS")
$ SHOW SYMBOL RADCPU
  0,0,0,1,1,4,1,5
```

   This example uses the system parameter RAD_CPUS as an argument for the F$GETSYI function. This argument returns a list of RAD/CPU pairs, separated by commas. In this example, the first RAD/CPU pair is 0,0 the second pair is 0,1 and so forth.

   RAD is supported on AlphaServer GS series systems and starting from OpenVMS Version 8.4, support is extended to NUMA capable Integrity servers.

F$IDENTIFIER

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. 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.

Any identifier holding the Name Hidden attribute will cause the F$IDENTIFIER to return an error when you do not hold the identifier in question or do not have access to the rights database. For further information on the attribute, see the relevant section in the *VSI OpenVMS Guide to System Security* [https://docs.vmssoftware.com/vsi-openvms-guide-to-system-security/#NAME_HID_ATT_SUB1].

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

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.

Arguments

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 because 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

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.

Arguments

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$LICENSE

F$LICENSE — Checks whether the specified license is loaded on the system.

Format

```
F$LICENSE(license-name[,producer-name])
```

Return Value

A character string stating TRUE or FALSE.

Arguments

license-name

Specifies the name of the license for which you want to check the status.

producer-name

Specifies the name of the company that produced the license. By default, DEC is assumed to be the producer on Alpha systems and HP is assumed to be the producer on Integrity server systems. To find an exception, specify a different producer name.

Examples

1. ```
   $ SHOW LICENSE VMSCLUSTER*
   Active licenses on node NODE1:
   ------- Product ID -------- ---- Rating ----- -- Version --
   Product            Producer Units Avail Activ Version Release
    Termination
   VMSCLUSTER         DEC          0  0     100    0.0  (none)    14-
   MAY-2005
   VMSCLUSTER-CLIENT  DEC          0  0     100    0.0  (none)    14-
   MAY-2005
   $ WRITE SYS$OUTPUT F$LICENSE("VMSCLUSTER")
   TRUE
   $ WRITE SYS$OUTPUT F$LICENSE("NONEXISTENT_PAK")
   FALSE
   ```

   In this example, the F$LICENSE function returns TRUE, which verifies that the VMSCLUSTER license is loaded on the system. In contrast, the status of hypothetical license NONEXISTENT_PAK is shown to be FALSE, indicating that it is not loaded on the system.

2. ```
   $ WRITE SYS$OUTPUT F$LICENSE("ABC")
   FALSE
   $ WRITE SYS$OUTPUT F$LICENSE("ABC","XYZ")
   TRUE
   ```

   In the first instance, no license for product ABC is found from the default producer (DEC or HP). In the second instance, an ABC PAK is found for producer XYZ.

F$LOCATE

F$LOCATE — Locates a specified portion of a character string and returns as an integer the offset of the first character. An offset is the position of a character or a substring relative to the beginning of the string. The first character in a string is always offset position 0 from the beginning of the string. If the substring is not found, F$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$MATCH_WILD

F$MATCH_WILD — Performs a wildcard matching between a candidate and a pattern string. TRUE is returned if the strings match.

Format

```
F$MATCH_WILD(candidate, pattern)
```

Arguments

candidate

A string to which the pattern string is compared.

pattern

A string on which a wildcard match is performed comparing the pattern to the candidate string.

Example

1. ```
   $ write sys$output f$match_wild ("This is a candidate","*c%%d*")
   TRUE
   $
   ```

   This command performs a wildcard match between the candidate *candidate* and pattern `*c%%d*` and found that the strings match.

2. ```
   $ write sys$output f$match_wild ("This is a candidate text", "*candi*)
   TRUE
   $
   ```

   This command checks to see if the pattern `candi` appears in the candidate.

F$MESSAGE

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[,message-component-list])
```

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 *OpenVMS System Messages: Companion Guide for Help Message Users*.

Arguments

status-code

Specifies the status code for which you are requesting error message text. You must specify the status code as an integer expression.

message-component-list

Specifies the system message component for which information is to be returned. If this parameter is null or unspecified, then all system message components are returned.

Table 12, "F$MESSAGE Keywords" describes the valid system message component keywords:

Table 12. F$MESSAGE Keywords

Component Keyword	Information Returned
FACILITY	Facility name
SEVERITY	Severity level indicator
IDENT	Abbreviation of message text
TEXT	Explanation of message

Note that when the FACILITY, SEVERITY, and IDENT code keywords are specified (individually or in combination), the resulting message code is prefaced with the percent (%) character. The individual parts of the message code are separated by hyphens when multiple code keywords are specified.

When only the TEXT keyword is specified, the resulting text is not prefaced with any character. When the TEXT keyword is specified with the FACILITY, SEVERITY, or IDENT code keyword, the message code is separated from the text by a combination of a comma and a blank (,).

Example

1.
```
$ 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.

2.
```
$ SUBMIT IMPORTANT.COM
$ SYNCHRONIZE /entry='$ENTRY'
$ IF $STATUS THEN EXIT
$!
$ JOB_STATUS = $STATUS
$!
$ IF "%JOBDELETE" .EQS. F$MESSAGE (JOB_STATUS, "IDENT")
$ THEN
  .
  .
  .
$ ELSE
$    IF "%JOBABORT" .EQS. F$MESSAGE (JOB_STATUS, "IDENT")
$    THEN
  .
```

```
   .
   .
$     ELSE
   .
   .
   .
$     ENDIF
$ ENDIF
   .
   .
   .
```

This command procedure submits a batch job and waits for it to complete. Upon successful completion, the procedure exits. If the job completes unsuccessfully, more processing is done based on the termination status of the batch job.

The first command submits the command procedure `IMPORTANT.COM`. In the second command, the **`SYNCHRONIZE`** command tells the procedure to wait for the job to finish. The third command determines if the job completed successfully and, if so, the procedure exits. The next command saves the status in a symbol.

The first **`IF`** statement uses `F$MESSAGE` to determine whether the job was deleted before execution. If so, it does some processing, possibly to resubmit the job or to inform a user via MAIL.

The next **`IF`** statement uses **`F$MESSAGE`** to determine whether the job was deleted during execution. As a result, some cleanup or human intervention may be required, which would be done in the **`THEN`** block.

If neither **`IF`** statement was true, then some other unsuccessful status was returned. Other processing, which would be done in the block following the **`ELSE`** statement, might be required.

F$MODE

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$MULTIPATH

F$MULTIPATH — Returns a specified item of information for a specific multipath-capable device.

Format

F$MULTIPATH(*device-name*,*item*,*context-symbol*)

Return Value

A character string containing the requested information.

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$MULTIPATH 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. Currently, the only valid item is MP_PATHNAME, which returns a string with the path name for the specified multipath-capable device.

context-symbol

Prior to the first use of F$MULTIPATH with MP_PATHNAME, the context symbol must be initialized to a value of 0. The F$MULTIPATH function is responsible for maintaining the value of the context symbol.

Caution

Do not modify the context symbol value after it has been initialized to 0; doing so could result in unpredictable behavior of F$MULTIPATH.

Description

Invokes the $DEVICE_PATH_SCAN system service to return a specified item of information for a specific multipath-capable device.

The F$MULTIPATH lexical function also returns any error messages generated by the $DEVICE_PATH_SCAN system service. For more information about the $DEVICE_PATH_SCAN system service, see the relevant section in the *VSI OpenVMS System Services Reference Manual* [https://docs.vmssoftware.com/vsi-openvms-system-services-reference-manual-a-getuai/#_4527DEVICE_PATH_SCAN].

Example

```
$    XYZ = 0
$
$LOOP:
$     PATH = F$MULTIPATH( "$1$DGA12", "MP_PATHNAME", XYZ )
$     IF PATH .EQS. "" THEN GOTO EXIT
$     WRITE SYS$OUTPUT "PATH NAME = ''PATH'"
$     GOTO LOOP
$
$EXIT:
$     EXIT
```

This example shows the use of `F$MULTIPATH` with the `MP_PATHNAME` item code. Note that the context symbol `XYZ` has been initialized to `0` outside of the loop. The output from this command procedure is shown below. When all paths for a given multipath device have been returned, the end of the list is signaled by the return of a blank path name.

```
path name = PGA0.5000-1FE1-0001-5782
path name = PGA0.5000-1FE1-0001-5783
path name = PGA0.5000-1FE1-0001-5781
path name = PGA0.5000-1FE1-0001-5784
path name = MSCP
```

F$PARSE

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. For more information, see the Description section for this lexical function.

In most cases, the F$PARSE function returns a null string ("") if an error is detected during the parse. For example, a null string is returned if the file specification has incorrect syntax or if a disk or directory does not exist, making the file specification logically incorrect. However, when you specify a field name or the SYNTAX_ONLY parse type, F$PARSE returns the appropriate information.

Arguments

filespec

Specifies a character string containing the file specification to be parsed.

The file specification can contain the asterisk (*) and the percent sign (%) wildcard characters. 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:

Keyword	Description
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 relevant section in the *VSI OpenVMS Record Management Services Reference Manual* [https://docs.vmssoftware.com/vsi-openvms-record-management-services-reference-manual/#PARSE_SERVICE_ROUTINE].

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$PARSE function supplies defaults, first from the *default-spec* argument and second from the *related-spec* argument. Note that the version number is not picked up from the *related-spec* argument. If names are not provided by these arguments, the F$PARSE function returns a null specification for these fields.

The parse operation simply validates that the provided file specification is syntactically correct; it does not enforce file specification semantics. For example, fields such as the version number are verified to contain five or fewer numeric digits, optionally preceded by a hyphen (-), but are not range checked. File specification semantics are enforced by services such as Open and Create.

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

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.

Arguments

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 o 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 mixed-architecture OpenVMS Cluster system.

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

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.

Arguments

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 *VSI OpenVMS Guide to System Security* [https://docs.vmssoftware.com/vsi-openvms-guide-to-system-security/]. 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, GROUP, and TMPMBX privileges and if you do not have 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

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$READLINK

F$READLINK — Returns the target file name which is indicated by the specified symbolic link, or NULL if the input is not a symbolic link.

Format

F$READLINK(*filespec*)

Return Value

Either the text contents of a symbolic link or NULL if the input is not a symbolic link.

Arguments

filespec

Specifies the name of the symbolic link file. You must specify the file name as a character string expression.

Example

1.
```
$ CREATE FILE.TXT
HELLO EXIT
$
$ CREATE/SYMLINK="FILE.TXT" L.LNK
$ X=F$READLINK("L.LNK")
$ SH SYM X
  X = "FILE.TXT"
$
```

   This example uses the F$READLINK function to return the target file name for a specified symbolic link.

2.
```
$ CREATE FILE.TXT
HELLO EXIT
```

```
$
$ CREATE/SYMLINK="FILE.TXT" L.LNK
$ CREATE/SYMLINK="L.LNK" LINK_TO_LINK.LNK
$ X= F$READLINK("LINK_TO_LINK.LNK")
$ SH SYM X
X = "L.LNK"
```

This example shows how F$READLINK functions when the target is a symbolic link file.

F$SEARCH

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$SEARCH function does not supply defaults for a file name or type. If the version is omitted, the specification for the file with the highest version number is returned. If the *filespec* argument contains the asterisk (*) or the percent sign (%) wildcard characters, each time F$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 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 relevant section in the *VSI OpenVMS Record Management Services Reference Manual* [https://docs.vmssoftware.com/vsi-openvms-record-management-services-reference-manual/#SEARCH_SERVICE_ROUTINE].

You can use the F$SEARCH function in a loop in a command procedure to return file specifications for all files that match a *filespec* argument containing an asterisk (*) or a percent sign (%) wildcard character. Each time the F$SEARCH function is executed, it returns the next file specification that matches the file specification that contains a wildcard character. 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 an asterisk (*) or the percent sign (%) wildcard characters 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 in one of the following ways:

- Explicitly, by stating a *stream-id* argument
- Implicitly, by omitting the *stream-id* argument and by using the same *filespec* argument each time you execute the F$SEARCH function

If you do not maintain the context of the search stream, you start a new search at the beginning of the directory file each time you specify a different *filespec* argument.

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$SYSTEM` directory. Only the latest version is returned because an asterisk (*) wildcard character is not used as the version number. The *filespec* argument `SYS$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 `.COM`. When it finds a `.COM` file, a pointer is set to maintain the search context. When the `F`

$SEARCH function is used the second time, it again starts searching from the top of the directory file for a file with a type .DAT. When the procedure loops back to the label START, the *stream-id* argument allows F$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.

3.
```
$ FILESPEC = F$SEARCH("TRNTO""SMITH SALLY""::DKA1:[PROD]*.DAT")
$ SHOW SYMBOL FILESPEC
  FILESPEC = "TRNTO"smith password"::DKA1:[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

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.

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.

Arguments

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 *VSI OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-openvms-user-s-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.

Note

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 *VSI OpenVMS System Services Reference Manual: GETUTC–Z* [https://docs.vmssoftware.com/vsi-openvms-system-services-reference-manual-getutc-z/#JUN_514].

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, see 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.

2.
```
$ SHOW PROCESS/PRIVILEGE

05-JUN-2001 15:55:09.60   RTA1:              User: HELRIEGEL

Process privileges:

Process rights identifiers:
 INTERACTIVE
 LOCAL

$ NEWPRIVS = F$SETPRV("ALL, NOOPER")
$ SHOW SYMBOL NEWPRIVS
  NEWPRIVS = "NOCMKRNL,NOCMEXEC,NOSYSNAM,NOGRPNAM,NOALLSPOOL,
      NOIMPERSONATE,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

05-JUN-2001 10:21:18.32   User: INAZU      Process ID: 00000F24
                          Node: TOKNOW     Process name: "_FTA23:"

Authorized privileges:
 NETMBX    SETPRV    SYSPRV    TMPMBX

Process privileges:
 ACNT                 may suppress accounting messages
 ALLSPOOL             may allocate spooled device
 ALTPRI               may set any priority value
 AUDIT                may direct audit to system security audit log
 BUGCHK               may make bug check log entries
 BYPASS               may bypass all object access controls
 CMEXEC               may change mode to exec
 CMKRNL               may change mode to kernel
```

```
 DIAGNOSE            may diagnose devices
 DOWNGRADE           may downgrade object secrecy
 EXQUOTA             may exceed disk quota
 GROUP               may affect other processes in same group
 GRPNAM              may insert in group logical name table
 GRPPRV              may access group objects via system protection
 IMPERSONATE         may impersonate another user
 IMPORT              may set classification for unlabeled object
 LOG_IO              may do logical i/o
 MOUNT               may execute mount acp function
 NETMBX              may create network device
 OPER                may perform operator functions
 PFNMAP              may map to specific physical pages
 PHY_IO              may do physical i/o
 PRMCEB              may create permanent common event clusters
 PRMGBL              may create permanent global sections
 PRMMBX              may create permanent mailbox
 PSWAPM              may change process swap mode
 READALL             may read anything as the owner
 SECURITY            may perform security administration functions
 SETPRV              may set any privilege bit
 SHARE               may assign channels to non-shared devices
 SHMEM               may create/delete objects in shared memory
 SYSGBL              may create system wide global sections
 SYSLCK              may lock system wide resources
 SYSNAM              may insert in system logical name table
 SYSPRV              may access objects via system protection
 TMPMBX              may create temporary mailbox
 UPGRADE             may upgrade object integrity
 VOLPRO              may override volume protection
 WORLD               may affect other processes in the world

Process rights:
 INTERACTIVE
 LOCAL

System rights:
 SYS$NODE_TOKNOW

$ NEWPRIVS = F$SETPRV(NEWPRIVS)
$ SHOW PROCESS/PRIVILEGE

05-JUN-2001 16:05:07.23   RTA1:             User: JERROM

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, VSI 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

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.

Arguments

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$SYMLINK_ATTRIBUTES

F$SYMLINK_ATTRIBUTES — Returns attribute information for a specified symbolic link file. Note that F$SYMLINK_ATTRIBUTES returns information for a specified symbolic link file, and not the target of the symbolic link. For information about the target file of a symbolic link, use F$FILE_ATTRIBUTES.

Format

F$SYMLINK_ATTRIBUTES(*filespec*,*item*)

Return Value

Either an integer or a character string, depending on the item you request.

Arguments

filespec

Specifies the name of the symlink file for which you are requesting information. You must specify the file name as a character string expression.

Only one file name can be specified at a time and wildcard characters are not allowed.

item

Indicates the attribute of the symlink file that must be returned. The item argument must be specified as a character string expression, and can be any one of the OpenVMS RMS field names listed in the following table:

Item	Type	Information Returned
AI	String	TRUE if after-image (AI) journaling is enabled; FALSE if disabled.
ALQ	Integer	Allocation quantity.
BDT	String	Backup date/time.
BI	String	TRUE if before-image (BI) journaling is enabled; FALSE if disabled.
BKS	Integer	Bucket size.
BLS	Integer	Block size.
CBT	String	TRUE if contiguous-best-try; otherwise FALSE.
CDT	String	Creation date/time.
CTG	String	TRUE if contiguous; otherwise FALSE.
DEQ	Integer	Default extension quantity.
DID	String	Directory ID string.
DIRECTORY	String	Returns TRUE or FALSE. Returns TRUE if it is a directory.
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; otherwise FALSE.
FFB	Integer	First free byte.
FID	String	File ID string.
FILE_LENGTH_HINT	String	Record count and data byte count in the form (*n*,*m*), where *n* is the record count and *m* is the data byte count. An invalidated count is specified by a -1 for *n* or *m*.
FSZ	Integer	Fixed control area size.
GBC	Integer	Global buffer count.
GBC32	Integer	Enhanced longword version of global buffer count with a per-file maximum size of about 2.1 billion for indexed files.
GBCFLAGS	String	Per-file management flags for sizing of global buffer cache. Returns PERCENT if global buffer count is expresses as a percent, DEFAULT if global

Item	Type	Information Returned
		buffer size is determined at runtime by an algorithm using two global buffer SYSGEN parameters (GB_CACHEALLMAX and GB_DEFPERCENT); or NONE if no per-file management flags are enabled for the file.
GRP	Integer	Owner group number.
JOURNAL_FILE	String	TRUE if the file is a journal; otherwise FALSE.
KNOWN	String	Known file; returns TRUE or FALSE to indicate whether file is installed with the Install utility (INSTALL). However, returns NOSUCHFILE if a file does not exist (for example, the file has been installed but subsequently deleted).
LOCKED	String	TRUE if a file is deaccessed-locked; otherwise FALSE.
LRL	Integer	Longest record length.
MBM	Integer	Owner member number.
MOVE	String	TRUE if movefile operations are enabled; otherwise FALSE.
MRN	Integer	This is not applicable.
MRS	Integer	Maximum record size.
NOA	Integer	This is not applicable.
NOBACKUP	String	FALSE if the file is marked for backup; TRUE if the file is marked NOBACKUP.
NOK	Integer	This is not applicable.
ORG	String	File organization; returns SEQ, REL, IDX.
PRESHELVED	String	TRUE if the file is preshelved; otherwise FALSE.
PRO	String	File protection string.
PVN	Integer	This is not applicable.
RAT	String	Record attributes; returns CR, PRN, FTN, "".
RCK	String	TRUE if read check; otherwise FALSE.
RDT	String	Revision date/time.

Item	Type	Information Returned
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.
RVN	Integer	Revision number.
SHELVABLE	String	TRUE if the file is shelvable; otherwise FALSE.
SHELVED	String	TRUE if the file is shelved; otherwise FALSE.
STORED_SEMANTICS	String	ASCII string that represents stored semantics.
UIC	String	Owner user identification code (UIC) string.
VERLIMIT	Integer	Version limit number. The value 32767 indicates that no version limit was set.
WCK	String	TRUE if write check; otherwise FALSE.

Example

```
$ DIR/LINK/FILE
  Directory SYS$SYSDEVICE:[EXAMPLES]

  TARGET.TXT;1                (5601,12,0)
  FOO.LNK;1 -> TARGET.TXT     (5600,12,0)
  Total of 2 files.

  $ FID_FILE = F$FILE_ATTRIBUTES("FOO.LNK","FID")
  $ FID_SYM  = F$SYMLINK_ATTRIBUTES("FOO.LNK","FID")

  $ SHOW SYMBOL FID_FILE
  FID_FILE = "(5601,12,0)"

  $ SHOW SYMBOL FID_SYM
  FID_SYM = "(5600,12,0)"
```

This example uses the F$SYMLINK_ATTRIBUTES function to return information about the symbolic link file FOO.LNK.

F$TIME

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

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.

A logical name for a logical name table must be defined in one of the following logical name tables:

- LNM$SYSTEM_DIRECTORY
- LNM$PROCESS_DIRECTORY

Note

If you subsequently create a table using the **CREATE/NAME_TABLE** command and want to make your private table accessible for F$TRNLNM, you must redefine one of the table logical names to include your private table. To see all the tables that are normally searched by F$TRNLNM, issue the following command:

```
$ SHOW LOGICAL/STRUCTURE LNM$DCL_LOGICAL
```

For more information, see the **CREATE/NAME_TABLE** command. See also the **SHOW LOGICAL** command in the *VSI OpenVMS DCL Dictionary: N–Z* [https://docs.vmssoftware.com/vsi-openvms-dcl-dictionary-n-z/#BRASS_112].

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 of 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 he 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 translation to be performed. The case argument controls both the case of the translation and whether the translation is to be interlocked or noninterlocked.

You can specify the case argument as any combination of CASE_BLIND(default), CASE_SENSITIVE, NONINTERLOCKED (default), and INTERLOCKED.

If the translation is case blind, the F$TRNLNM searches the logical name table for the first occurrence of the logical name, regardless of the case, and returns the translation. If no match is found for either case, the function returns a null string ("").

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 ("").

If the translation is interlocked, the F$TRNLNM function does not take effect until all clusterwide logical name modifications in progress complete. Then, if a match is found, the result of the translation is returned. If no match is found, the F$TRNLNM function returns a null string ("").

If the translation is noninterlocked, the F$TRNLNM function takes effect immediately. If a match is found, the result of the translation is returned. If no match is found, the F$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.
CLUSTERWIDE	String	TRUE or FALSE to indicate whether the logical name is in a clusterwide name table.
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,

Item	Return Type	Information Returned
		using the CRELOG attribute, then TRUE is returned. Otherwise, FALSE is 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.

4. `$ foo=f$trnlnm("FOO","LNM$SYSCLUSTER",,,"INTERLOCKED",)`

   In this example, logical name FOO is translated in the LNM$SYSCLUSTER table in an interlocked manner; that is, all clusterwide logical name modifications in progress on this and other nodes are completed before the translation occurs. This ensures that the translation is based on the most recent definition of FOO.

   Because the case translation is not specified, the translation is by default CASE_BLIND.

5. `$ foo=f$trnlnm("FOO","LNM$SYSCLUSTER",,,"INTERLOCKED,CASE_SENSITIVE",)`

   This example specifies both case sensitive and interlocked translation.

F$TYPE

F$TYPE — Returns the data type of a symbol. 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. The string
   ```

STRING is returned if the symbol is equated to a character string whose characters do not form a valid integer. If the symbol is undefined, a null string ("") is returned.

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 13, "Context Symbol Types"*.

Table 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.

Arguments

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$UNIQUE

F$UNIQUE — Generates a string that is suitable to be a file name and is guaranteed to be unique across the cluster. Unique file names can be useful when creating temporary files. See **CLOSE/DISPOSITION** for an example. The F$UNIQUE function has no arguments, but must be followed by a blank pair of parentheses.

Format

```
F$UNIQUE()
```

Return Value

A character string containing the unique string.

Arguments

None.

Examples

1.
```
$ WRITE SYS$OUTPUT F$UNIQUE()
41485355524115971 1D7DF797CCF573F
$
$ WRITE SYS$OUTPUT F$UNIQUE()
41485355524150981 1D7DF797E3F2777
```

```
$
```

This example shows how a unique string is returned on subsequent **WRITE** commands.

2.
```
$ OPEN/WRITE TEMP_FILE 'F$UNIQUE()
$ DIRECTORY
Directory WORK1:[TEST]
594B53554C421C9C11D75463D61F58B7.DAT;1
Total of 1 file.
$
$ CLOSE/DISPOSITION=DELETE TEMP_FILE
$ DIRECTORY
%DIRECT-W-NOFILES, no files found
$
```

The first command creates a temporary file and gives it a unique name, which is displayed by the subsequent **DIRECTORY** command. After the file is later closed and deleted, it no longer shows up in the directory.

F$USER

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

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 the initial state of verification.

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.

Example

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.