VMS

digital

VMS Message Utility Manual

Order Number AA–LA63A–TE

VMS Message Utility Manual

Order Number: AA–LA63A–TE

April 1988

This manual describes the VMS Message Utility.

Revision/Update Information: This manual supersedes the *VAX/VMS Message Utility Reference Manual,* Version 4.0.

Software Version: VMS Version 5.0

digital equipment corporation
maynard, massachusetts

April 1988

DEC	DIBOL	UNIBUS
DEC/CMS	EduSystem	VAX
DEC/MMS	IAS	VAXcluster
DECnet	MASSBUS	VMS
DECsystem-10	PDP	VT
DECSYSTEM-20	PDT	
DECUS	RSTS	
DECwriter	RSX	digital™

ZK4550

HOW TO ORDER ADDITIONAL DOCUMENTATION
DIRECT MAIL ORDERS

USA & PUERTO RICO*	**CANADA**	**INTERNATIONAL**
Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061	Digital Equipment of Canada Ltd. 100 Herzberg Road Kanata, Ontario K2K 2A6 Attn: Direct Order Desk	Digital Equipment Corporation PSG Business Manager c/o Digital's local subsidiary or approved distributor

In Continental USA and Puerto Rico call 800-258-1710.
In New Hampshire, Alaska, and Hawaii call 603-884-6660.
In Canada call 800-267-6215.
*Any prepaid order from Puerto Rico must be placed with the local Digital subsidiary (809-754-7575).
Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Westminster, Massachusetts 01473.

Production Note

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

™ PostScript is a trademark of Adobe Systems, Inc.

Contents

Preface

Intended Audience

This document is intended for programmers and general users of the VMS operating system.

Document Structure

This document consists of the following five sections:

- Description—Provides a full description of the Message Utility (MESSAGE).
- Usage Summary—Outlines the following information:

  –Invoking the utility
  –Exiting the utility
- Qualifiers—Describes MESSAGE qualifiers, including format, parameters, and examples.
- Commands—Describes MESSAGE source file statements including format, parameters, and examples.
- Examples—Provides examples for using message files and pointer files.

Associated Documents

The *VMS Linker Utility Manual* contains information about linking object modules and creating executable, nonexecutable, and shareable images.

Conventions

Convention	Meaning
$\boxed{\text{RET}}$	In examples, a key name (usually abbreviated) shown within a box indicates that you press a key on the keyboard; in text, a key name is not enclosed in a box. In this example, the key is the RETURN key. (Note that the RETURN key is not usually shown in syntax statements or in all examples; however, assume that you must press the RETURN key after entering a command or responding to a prompt.)
CTRL/C	A key combination, shown in uppercase with a slash separating two key names, indicates that you hold down the first key while you press the second key. For example, the key combination CTRL/C indicates that you hold down the key labeled CTRL while you press the key labeled C. In examples, a key combination is enclosed in a box.
$ SHOW TIME 05-JUN-1988 11:55:22	In examples, system output (what the system displays) is shown in black. User input (what you enter) is shown in red.
$ TYPE MYFILE.DAT . . .	In examples, a vertical series of periods, or ellipsis, means either that not all the data that the system would display in response to a command is shown or that not all the data a user would enter is shown.
input-file, . . .	In examples, a horizontal ellipsis indicates that additional parameters, values, or other information can be entered, that preceding items can be repeated one or more times, or that optional arguments in a statement have been omitted.
[logical-name]	Brackets indicate that the enclosed item is optional. (Brackets are not, however, optional in the syntax of a directory name in a file specification or in the syntax of a substring specification in an assignment statement.)
quotation marks apostrophes	The term quotation marks is used to refer to double quotation marks ("). The term apostrophe (') is used to refer to a single quotation mark.

New and Changed Features

This version of the Message Utility (MESSAGE) includes no significant technical changes.

MESSAGE Description

This section describes how to use the Message Utility (MESSAGE).

1 Message Format

Messages are displayed as a line of alphanumeric codes. The text of the message explains the condition that caused the message to be displayed.

Messages are displayed in the following format:

```
%FACILITY-L-IDENT, message-text
```

FACILITY

Specifies the abbreviated name of the software component that issued the message.

L

Shows the severity level of the condition that caused the message. The five severity levels are represented by the following codes:

S	Success
I	Informational
W	Warning
E	Error
F	Fatal or severe

IDENT

Identifies a symbol of up to 15 characters that represents the message.

message-text

Explains the cause of the message. The message text can include up to 255 formatted-ASCII-output (FAO) arguments. For example, an FAO argument can be used to display the instruction where an error occurred or a value that you should be aware of.

% and ,

Included as delimiters if any of the first three fields—FACILITY, L, or IDENT—are present.

If you suppress FACILITY, L, and IDENT, the first character of the message text is capitalized by the Put Message ($PUTMSG) system service.

The following is a typical message:

```
%TYPE❶-W-❷OPENIN❸, error opening _DBO:[ROSE]STATS.FOR;❹ as input❺
```

❶ TYPE is the facility.

❷ W, warning, is the severity level.

❸ OPENIN is the IDENT.

❹ _DBO:[ROSE]STATS.FOR is the FAO argument.

❺ "Error opening _DBO:[ROSE]STATS.FOR; as input" is the message text.

2 Constructing Messages

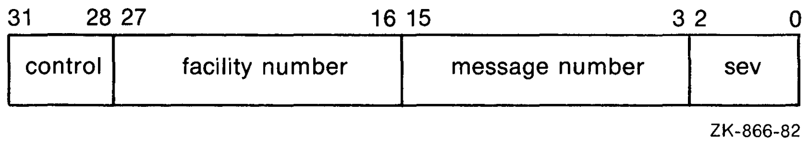
You construct messages by writing a message source file, by compiling it using the Message Utility, and by linking the resulting object module with your facility object module. When you run your program, the Put Message ($PUTMSG) system service finds the information to use in the message by using a message argument vector.

The message argument vector includes the message code, a 32-bit value that uniquely identifies the message. The message code, which is created from information defined in the message source file, consists of the following:

- The severity level defined in the severity directive or message definition
- The message number assigned automatically by a message definition or specified with the base message number directive
- The facility number defined in the facility directive
- Internal control flags

Figure MSG–1 shows the arrangement of the bits in the message code.

Figure MSG–1 Message Code



You can refer to the message code in your programs by means of a global symbol called the message symbol, which also is defined by information from the message source file. The message symbol, which appears in the compiled message file, consists of the following:

- The symbol prefix defined in the facility directive
- The symbol name defined in the message definition

2.1 The Message Source File

The message source file consists of message definition statements and directives that define the message text, the message code values, and the message symbol. The various elements that can be included in a message source file are as follows:

- Facility directive
- Severity directive
- Base message number directive
- Message definition
- Literal directive
- Identification directive
- Listing directives
- End directive

Usually, the first statement in a message source file is a .TITLE directive, which allows you to specify a module name for the compiled message file. You must specify a .FACILITY directive after the .TITLE directive. All the messages defined after a .FACILITY directive are associated with that facility. A .END directive or a new .FACILITY directive ends the list of messages associated with a particular facility.

You must define a severity level for each message either by specifying a .SEVERITY directive or by including a severity qualifier as part of the message definition.

Each message defined in the message source file must have a facility and a message definition associated with it. All other message source file statements are optional. See the Source File Statements Section for a detailed description of the format of each of these message source file statements.

The TESTMSG.MSG file that follows is a sample message source file. The messages for the associated FORTRAN program, TEST.FOR, are defined in TESTMSG.MSG with the following lines:

```
 .FACILITY       TEST,1 /PREFIX=MSG_
 .SEVERITY       ERROR
SYNTAX          <Syntax error in string '!AS'>/FAO=1
ERRORS          <Errors encountered during processing>
 .END
```

The FORTRAN program, TEST.FOR, contains the following lines:

```
EXTERNAL MSG_SYNTAX,MSG_ERRORS
CALL LIB$SIGNAL(MSG_SYNTAX,%VAL(1),'ABC')
CALL LIB$SIGNAL(MSG_ERRORS)
END
```

In addition to defining the message data, TESTMSG.MSG also defines the message symbols MSG_SYNTAX and MSG_ERRORS that are included as arguments in the procedure calls of TEST.FOR. The function %VAL is a required FORTRAN compile-time function. The first call also includes the string **ABC** as an FAO argument.

2.2 Compiling the Message Source File

You must compile message source files into object modules before you can use the messages defined in them. You compile your message source file by typing the MESSAGE command followed by the file specification of the message source file. For example:

```
$ MESSAGE TESTMSG
```

This command compiles the message source file, TESTMSG.MSG, and creates an object module file, TESTMSG.OBJ.

For your convenience, you can put message object modules into object module libraries, which you can then link with facility object modules.

2.3 Linking the Message Object Module

After you compile the message file, you must link the message object module with the facility object module (created when the source file was compiled) to produce one executable image file.

For example, you link the message object module, TESTMSG.OBJ, to the FORTRAN object module, TEST.OBJ, to create the executable program, TEST.EXE, with the following command:

```
$ LINK/NOTRACE TEST+TESTMSG
```

At this point, you can execute the program, which contains both the message data and the facility code, with the following command:

```
$ RUN TEST
```

If an error occurs when you execute the program, the following messages are displayed:

```
%TEST-E-SYNTAX, Syntax error in string ABC
%TEST-E-ERRORS, Errors encountered during processing
```

3 Using Message Pointers

Message pointers are generally used when you need to provide different message texts for the same set of messages, for example, a multilingual situation. When you use message pointers, you do not link the message object module directly with the facility object module. Consequently, you do not have to relink the executable image file to change the message text included in it.

To use a pointer, you must create a nonexecutable message file that contains the message text, and a pointer file that contains the symbols and pointer to the nonexecutable file.

You create the nonexecutable message file by compiling and linking a message source file. For example, to create the nonexecutable message file COBOLMF.EXE, you first create the object module by compiling the message source file, COBOLMSG.MSG, with the following command:

```
$ MESSAGE/NOSYMBOLS COBOLMSG
```

You link the resulting object module with the following command:

```
$ LINK/SHAREABLE=SYS$MESSAGE:COBOLMF COBOLMSG.OBJ
```

By default, the Linker places newly created images in your default device and directory. In the preceding example, the nonexecutable image COBOLMF.EXE is placed in the system message library SYS$MESSAGE.

You create the pointer file by recompiling the message source file with the MESSAGE/FILE_NAME command. To avoid confusion, the pointer file should have a different file name from the nonexecutable file.

The object module resulting from the MESSAGE/FILE_NAME command contains only global symbols and the file specification of the nonexecutable message file.

For example, the following command creates the object module MESPNTR.OBJ, which contains a pointer to the nonexecutable message file COBOLMF.EXE:

```
$ MESSAGE/FILE_NAME=COBOLMF /OBJECT=MESPNTR COBOLMSG
```

In addition to the pointers, the object module, MESPNTR.OBJ, contains the global symbols defined in the message source file, COBOLMSG.MSG. If the destination of the nonexecutable message file is not SYS$MESSAGE, you must specify the device and directory in the file specification for the /FILE_NAME qualifier.

After you create the pointer object module, you can then link it with the facility object module.

For example, the following command links the object module, MESPNTR.OBJ, to the COBOL program, COBOLCODE:

```
$ LINK COBOLCODE,MESPNTR
```

When you run the resulting facility image file, the $GETMSG system service retrieves the message data from the message file, COBOLMF.

Figure MSG–2 illustrates the relationship of the files in this example.

4 The SET MESSAGE Command

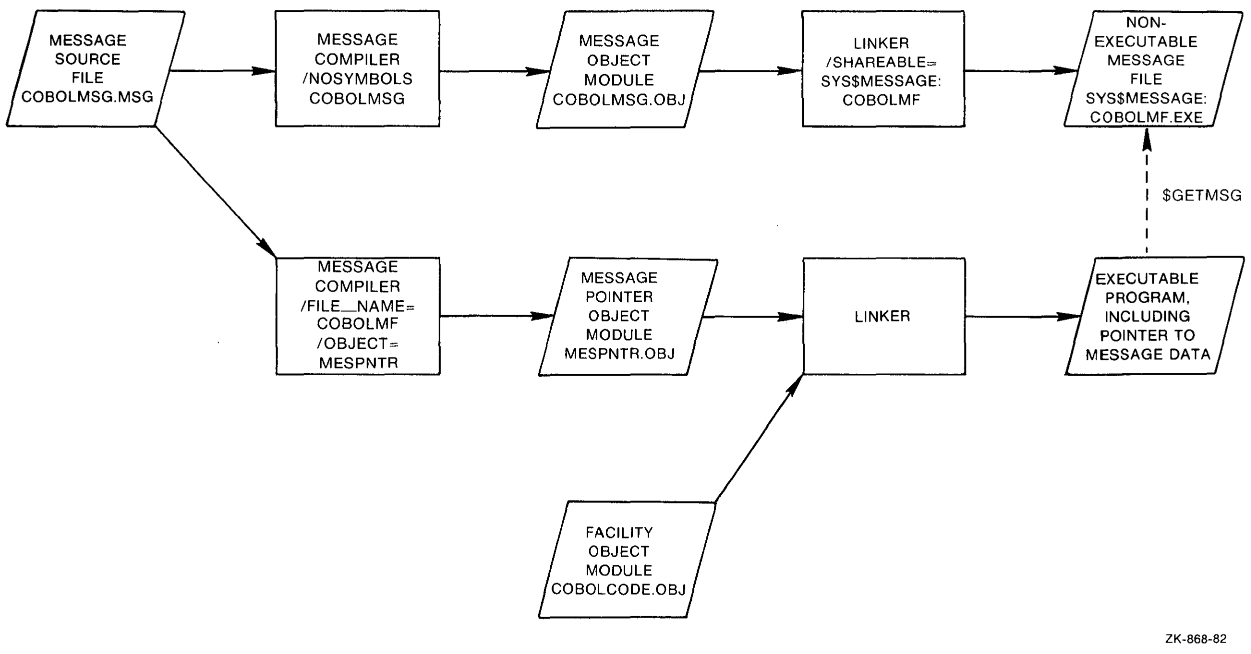
The SET MESSAGE command allows you to do the following:

- Suppress or enable the various fields of the messages in your process.
- Supplement the system message data with the message data in a nonexecutable message image for your process.

For example, the following SET MESSAGE command specifies that the message information in MYMSG.EXE supplements the existing system messages:

```
$ SET MESSAGE MYMSG
```

Figure MSG–2 Creating a Message Pointer


MESSAGE SOURCE FILE COBOLMSG.MSG
MESSAGE COMPILER /NOSYMBOLS COBOLMSG
MESSAGE OBJECT MODULE COBOLMSG.OBJ
LINKER /SHAREABLE= SYS$MESSAGE: COBOLMF
NON-EXECUTABLE MESSAGE FILE SYS$MESSAGE: COBOLMF.EXE
$GETMSG
MESSAGE COMPILER /FILE_NAME= COBOLMF /OBJECT= MESPNTR
MESSAGE POINTER OBJECT MODULE MESPNTR.OBJ
LINKER
EXECUTABLE PROGRAM, INCLUDING POINTER TO MESSAGE DATA
FACILITY OBJECT MODULE COBOLCODE.OBJ
ZK-868-82

In addition, the SET MESSAGE command used with one or more qualifiers suppresses or enables one or more fields in a messsage. For example, the following command suppresses the IDENT field in a message:

```
$ SET MESSAGE/NOIDENTIFICATION
```

For more information about the SET MESSAGE command, see the *VMS DCL Dictionary*.

5 Message Source Files

The message source file contains statements or directives and the information included in the message, the message code, and the message symbol.

Source File Statements

Message source file statements are embedded within a message source file. Generally, message source file statements help construct the message code and the message symbol, and control output listings. The message source file statements or directives are as follows:

- Facility directive .FACILITY
- Severity directive .SEVERITY
- Base message number directive .BASE
- Message definition message-name
- End directive .END
- Literal directive .LITERAL

- Identification directive .IDENT
- Listing directives
  - Title directive .TITLE
  - Page directive .PAGE

Many of these statements accept qualifiers and parameters. The specific format of each of the message source file statements is described in detail in the section on MESSAGE commands.

Any line in the message source file, except lines that contain the .TITLE directive, can include a comment delimited by an exclamation point. You may insert extra spaces and tabs in any line to improve readability.

The listing title specified with the .TITLE directive and the message text specified in the message definition must occupy only one line. All other statements in a message source file can occupy any number of lines; text that continues onto the next line must end with a hyphen.

Defining Symbols in the Message Source File

Symbols defined in the message source file can include any of the following characters:

A through Z
a through z
1 through 9
$ (dollar sign)
_ (underscore)

Using Expressions in the Message Source File

Expressions used in the message source file can include any of the following radix operators:

^X	Hexadecimal
^O	Octal
^D	Decimal

Radix operators specify the radix of a numeric value. The default radix is decimal.

Expressions can include symbols and the plus sign (+), which assigns a positive value, and minus sign (–), which assigns a negative value. Expressions can include the following binary operators:

+	Addition
–	Subtraction
*	Multiplication
/	Division
@	Arithmetic shift

Expressions can also include parentheses as grouping operators. Expressions enclosed in parentheses are evaluated first; nested parenthetical expressions are evaluated from inside to outside.

MESSAGE Usage Summary

The VMS Message Utility (MESSAGE) allows you to supplement the VMS system messages with your own messages. Your messages can indicate that an error has occurred. Messages can also indicate other conditions, for example, that a routine has run successfully, or that a default value has been assigned.

FORMAT

MESSAGE *file-spec[,...]*

COMMAND PARAMETER

file-spec

Specifies the message source file to be compiled. If you do not specify a file type, the default is MSG. Wildcard characters are allowed in the file specification or specifications.

If you specify more than one message source file, separated by either commas or plus signs, the files are concatenated and compiled as a single file.

If you specify SYS$INPUT, the message source file or files must immediately follow the MESSAGE command in the input stream, and both the object module name, identified by the /OBJECT qualifier, and the listing file name, identified by the /LIST qualifier, must be stated explicitly.

usage summary

The DCL command MESSAGE invokes the Message Utility. After compiling the message source file, the Message Utility returns you to DCL command level. For details about message statements and directives, qualifiers, and parameters in message source files, see the Message Commands Section.

MESSAGE QUALIFIERS

MESSAGE command qualifiers allow you to specify the type and contents of output files produced. In addition, MESSAGE command qualifiers allow you to create nonexecutable message files that contain pointers to files that contain message data. Output files produced by command qualifiers are named in accordance with the rules described in the *VMS DCL Dictionary*.

/FILE_NAME

Specifies whether the object module contains a pointer to a file containing message data.

FORMAT

/FILE_NAME=*file-spec*
/NOFILE_NAME

QUALIFIER VALUE

file-spec
Identifies a nonexecutable message file. The default device and directory for the file specification is SYS$MESSAGE, and the default file type is EXE. No wildcard characters are allowed in the file specification.

DESCRIPTION

The /[NO]FILE_NAME qualifier specifies whether the object module contains a pointer to a file containing message data. By default, the object module contains only compiled message information and no pointers.

The /FILE_NAME and /TEXT qualifiers cannot be used together because a message pointer file cannot contain message text. The message text is contained in the nonexecutable message file, specified by the /FILE_NAME qualifier.

EXAMPLES

1

```
$ MESSAGE COBOLMSG
```

This MESSAGE command creates the message object module, COBOLMSG.OBJ, by compiling the message source file, COBOLMSG.MSG. The default qualifier /NOFILE_NAME is implied.

2

```
$ MESSAGE/FILE_NAME=COBOLMF COBOLMSG
```

This MESSAGE command creates a message pointer file, COBOLMSG.OBJ, which contains a pointer to the nonexecutable message file, SYS$MESSAGE:COBOLMF.EXE.

/LIST

Controls whether an output listing is created, and optionally provides an output file specification for the listing.

FORMAT

/LIST*[=file-spec]*
/NOLIST

QUALIFIER VALUE

file-spec
Specifies an output file specification for the listing file. The default device and directory are the current device and directory. The default file type is LIS. No wildcard characters are allowed in the file specification.

DESCRIPTION

The /LIST qualifier creates a listing file. If you do not specify a file specification, the listing file has the same name as the first message source file being compiled and a file type of LIS. When you compile message source files in batch mode, the output listing is created by default; however, in interactive mode, the default is to produce no output listing.

EXAMPLE

```
$ MESSAGE/LIST=MSGOUTPUT  COBOLMSG
```

This MESSAGE command compiles the message source file COBOLMSG.MSG and creates the output listing MSGOUTPUT.LIS in your current directory.

/OBJECT

Controls whether an object module is created by the message compiler and optionally provides a file specification for the object module.

FORMAT

/OBJECT*[=file-spec]*
/NOOBJECT

QUALIFIER VALUE

file-spec
Specifies a file specification for the object module. The default device and directory are the current device and directory; no wildcard characters are allowed in the file specification.

DESCRIPTION

By default, the message compiler creates an object module that contains the message data. If you do not specify a file specification, the object module has the same name as the first message source file being compiled and a file type of OBJ.

EXAMPLES

1
```
$ MESSAGE COBOLMSG
```

This MESSAGE command creates the message object module, COBOLMSG.OBJ, by compiling the message source file, COBOLMSG.MSG. The default qualifier /OBJECT is implied.

2
```
$ MESSAGE/FILE_NAME=COBOLMF /OBJECT=MESPNTR COBOLMSG
```

This MESSAGE command creates the object module, MESPNTR.OBJ, which contains a pointer to the nonexecutable message file, COBOLMF.EXE.

/SYMBOLS

Controls whether global symbols are present in the object module. By default, object modules are created with global symbols.

FORMAT

/SYMBOLS
/NOSYMBOLS

QUALIFIER VALUES

None.

DESCRIPTION

By default, the message compiler creates an object module with global symbols. The /SYMBOLS qualifier requires that the /OBJECT qualifier be in effect, either explicitly or implicitly. If you are creating both a pointer object module and a nonexecutable message image, you can compile the object module, which becomes the nonexecutable image, with the /NOSYMBOLS qualifier. The symbols have to be only in the pointer object module.

EXAMPLE

```
$ MESSAGE/FILE_NAME=COBOLMF /OBJECT=MESPNTR/SYMBOLS COBOLMSG
```

This MESSAGE command creates the object module, MESPNTR.OBJ, which contains global symbols.

/TEXT

Controls whether the message text is present in the object module.

FORMAT

/TEXT
/NOTEXT

QUALIFIER VALUES

None.

DESCRIPTION

By default, the message compiler creates an object module that contains message text. The message text is obtained from the nonexecutable message file specified by the /FILE_NAME qualifier. The /TEXT and /FILE_NAME qualifiers cannot be used together because a message pointer file cannot contain message text.

The /TEXT qualifier requires that the /OBJECT qualifier be in effect, either explicitly or implicitly.

You can use the /NOTEXT qualifier with the /SYMBOLS qualifier to produce an object module containing only global symbols.

EXAMPLE

```
$ MESSAGE/FILE_NAME=COBOLMF/NOTEXT /OBJECT=MESPNTR COBOLMSG
```

This MESSAGE command creates the object module, MESPNTR.OBJ, which does not contain text; instead, it contains a pointer to the nonexecutable message file, COBOLMF.EXE.

MESSAGE COMMANDS

This section describes the message source file statements.

Base Message Number Directive

Defines the value used in constructing the message code.

FORMAT

.BASE *number*

PARAMETER

number
Specifies a message number to be associated with the next message definition, or an expression that is evaluated as the desired number.

QUALIFIERS

None.

DESCRIPTION

By default, all of the messages following a facility directive are numbered sequentially, beginning with 1.

If you need to supersede this default numbering system (for example, if you want to reserve some message numbers for future assignment), specify a message number of your choice using the base message number directive. The message number is used as a base for the sequential numbering of all messages that follow until either another .BASE directive or the end of the messages belonging to the facility is encountered.

EXAMPLE

```
 .TITLE         SAMPLE Error and Warning Messages
 .IDENT         'VERSION 4.00'
 .FACILITY      SAMPLE,1/PREFIX=ABC_  ❶
 .SEVERITY      ERROR

UNRECOG        < Unrecognized keyword !AS>/FAO_COUNT=1
AMBIG          < Ambiguous keyword>

 .SEVERITY      WARNING
 .BASE          10    ❷
SYNTAX         < Invalid syntax in keyword>

 .END
```

The facility number (facnum) in the facility statement ❶ defines the first two message numbers as 1 and 2. This sequential numbering is superseded by the base message number directive ❷ which assigns the message number 10 to the third message.

End Directive

Terminates the entire list of messages for the facility.

FORMAT

.END

PARAMETERS

None.

QUALIFIERS

None.

DESCRIPTION

An End directive terminates the entire list of messages for a facility. A .FACILITY directive also terminates a list of messages.

EXAMPLE

```
 .TITLE        SAMPLE Error and Warning Messages
 .IDENT        'VERSION 4.00'
 .FACILITY     SAMPLE,1/PREFIX=ABC_
 .SEVERITY     ERROR

UNRECOG       < Unrecognized keyword !AS>/FAO_COUNT=1
AMBIG         < Ambiguous keyword>

 .SEVERITY     WARNING
 .BASE         10
SYNTAX        < Invalid syntax in keyword>
 .END❶
```

The .END directive ❶ terminates the list of messages for the SAMPLE facility.

Facility Directive

Specifies the facility to which the messages apply.

FORMAT

.FACILITY *[/qualifier,...] facnam[,]facnum [/qualifier,...]*

PARAMETERS

facnam
Specifies the facility name used in the facility field of the message and in the symbol representing the facility number. The facility name can be up to nine characters.

facnum
Specifies the facility number used to construct the 32-bit value of the message code. A decimal value in the range 1 to 2047, or an expression that evaluates to a value in that range may be used. The system manager usually assigns facility numbers so that no two facilities have the same number.

QUALIFIERS

/PREFIX=prefix
Defines an alternate symbol prefix to be used in the message symbol for all messages referring to this facility. The default symbol prefix is the facility name followed by an underscore (_). If the /SYSTEM qualifier is also specified, the default prefix is the facility name followed by a dollar sign and an underscore ($_). The combined length of the prefix and the message symbol name cannot exceed 31 characters. The maximum length of an alternate symbol prefix created with the /PREFIX qualifier is nine characters.

/SHARED
Inhibits the setting of the facility-specific bit in the message code. The /SHARED qualifier is used only for system services and shared messages, and is reserved for DIGITAL use.

/SYSTEM
Inhibits the setting of the customer facility bit in the message code. This qualifier is reserved for DIGITAL use.

DESCRIPTION

The .FACILITY directive is the first directive in a message source file. All of the lines following a .FACILITY directive apply to that facility until an end statement or another facility statement is reached. You must specify the facility name and the facility number in a .FACILITY directive. The facility name and facility number can be separated by a comma or by any number of spaces or tabs.

The .FACILITY directive creates a global symbol of the following form:

```
facnam$_FACILITY
```

You can use this symbol to refer to the facility number assigned to the facility.

EXAMPLE

```
 .TITLE         SAMPLE Error and Warning Messages
 .IDENT         'VERSION 4.00'
 .FACILITY      SAMPLE,1/PREFIX=ABC_     ❶
 .SEVERITY      ERROR

UNRECOG        < Unrecognized keyword !AS>/FAO_COUNT=1
AMBIG          < Ambiguous keyword>

 .SEVERITY      WARNING
 .BASE          10
SYNTAX         < Invalid syntax in keyword>

 .END
```

The facility statement ❶ in this message source file defines the messages belonging to the facility (facnam) SAMPLE with a facility number (facnum) of 1. The message numbers begin with 1 and continue sequentially. The /PREFIX=ABC_ qualifier defines the message symbols ABC_UNRECOG, ABC_AMBIG, and ABC_SYNTAX.

Identification Directive

Identifies the object module the Message Utility produces.

FORMAT

.IDENT *string*

PARAMETER

string
Identifies the object module, for example, a string that identifies a version number. If it is not delimited, the string is a 1- to 31-character string of alphanumeric characters, underscores, and dollar signs. If other characters are used, then the string must be delimited with either apostrophes or quotation marks.

QUALIFIERS

None.

DESCRIPTION

The identification directive is used in addition to the name you assign to the module with the .TITLE directive. You can label the object module by specifying a character string with the directive. If a message source file contains more than one identification directive, the last directive establishes the character string that forms part of the object module identification.

EXAMPLE

```
 .TITLE         SAMPLE Error and Warning Messages
 .IDENT         'VERSION 4.00'❶
 .FACILITY      SAMPLE,1/PREFIX=ABC_
 .SEVERITY      ERROR

UNRECOG        <Unrecognized keyword !AS>/FAO_COUNT=1
AMBIG          <Ambiguous keyword>

 .SEVERITY      WARNING
 .BASE          10
SYNTAX         < Invalid syntax in keyword>

 .END
```

This identification directive ❶ identifies the object module that the Message Utility produces.

Literal Directive

Defines global symbols in your message source file. You can either assign values to these symbols or use the default values the directive provides.

FORMAT

.LITERAL *symbol[=value][,...]*

PARAMETERS

symbol
Specifies a symbol name.

value
Specifies any valid expression. If you omit the value, a default value is assigned. The default value is 1 for the first symbol in the directive and 1 plus the last value assigned for subsequent symbols.

QUALIFIERS

None.

DESCRIPTION

You can use the .LITERAL directive to define a symbol as the value of another previously defined symbol, or as an expression that results from operations performed on previously defined symbols.

EXAMPLES

1
```
 .LITERAL    A,B,C
```

The values of A, B, and C will be 1, 2, and 3.

2
```
 .FACILITY       SAMPLE,1/PREFIX=MSG$_
 .SEVERITY       ERROR
FIRST           < first error>
  .
  .
  .
LAST            < last error>
 .LITERAL        LASTMSG=MSG$_LAST   ❶

 .LITERAL        NUMSG=(MSG$_LAST@-3)-(MSG$_FIRST@-3)   !   of messages ❷
```

In this example, symbols defined in the facility and message definitions are used to assign values to symbols created with the .LITERAL directives.

The first .LITERAL directive ❶ defines a symbol that has the value of the last 32-bit message code defined. The second .LITERAL directive ❷ defines the total number of messages in the source file.

Message Definition

Defines the message symbol, the message text, and the number of FAO arguments that can be printed with the message.

FORMAT

name[/qualifier,...] <message-text> [/qualifier,...]

PARAMETERS

name

Specifies the name that is combined with the symbol prefix (defined in the .FACILITY directive) to form the message symbol. The combined length of the prefix and the message symbol name cannot exceed 31 characters.

The name is used in the IDENT field of the message unless you specify the /IDENTIFICATION qualifier in the message definition.

message-text

Defines the text explaining the condition that caused the message to be displayed. The message text can be delimited either by angle brackets or by quotation marks. The text can be up to 255 bytes long; however, you cannot continue the delimited text onto another line. The message text can include FAO directives that insert ASCII strings into the resulting message; the Formatted ASCII Output ($FAO) system service uses these directives. If you include an FAO directive, you must also use the /FAO_COUNT qualifier.

QUALIFIERS

/FAO_COUNT=n

Specifies the number of FAO arguments to be included in the message at execution time. The number specified must be a decimal number in the range 0 through 255. The $PUTMSG system service, when constructing the final message text, uses *n* to determine how many arguments are to be given to the $FAO service. The default value for *n* is 0.

/IDENTIFICATION=name

Specifies an alternate character string to be used as the IDENT field of the message. The name can include up to nine characters. If you do not specify this qualifier, the name defined in the message definition is used.

/USER_VALUE=n

Specifies an optional user value that can be associated with the message. The value must be a decimal number in the range of 0 through 255. The default is 0. The value can be retrieved by the Get Message ($GETMSG) system service for use in classifying messages by type or by action to be taken.

/SUCCESS

Specifies the success level for a message. This qualifier overrides any .SEVERITY directive in effect. If no .SEVERITY directive is in effect, you must use this qualifier to specify the success level.

/INFORMATIONAL

Specifies the informational level for a message. This qualifier overrides any .SEVERITY directive in effect. If no .SEVERITY directive is in effect, you must use this qualifier to specify the informational level.

/WARNING

Specifies the warning level for a message. This qualifier overrides any .SEVERITY directive in effect. If no .SEVERITY directive is in effect, you must use this qualifier to specify the warning level.

/ERROR

Specifies the error level for a message. This qualifier overrides any .SEVERITY directive in effect. If no .SEVERITY directive is in effect, you must use this qualifier to specify the error level.

/SEVERE

Specifies the severity level for a message. This qualifier overrides any .SEVERITY directive in effect. If no .SEVERITY directive is in effect, you must use this qualifier to specify the severity level.

/FATAL

Specifies the fatality level for a message. This qualifier overrides any .SEVERITY directive in effect. If no .SEVERITY directive is in effect, you must use this qualifier to specify the fatality level.

DESCRIPTION

The message definition specifies the message text that is to be displayed and the name used in the IDENT field of the message. Additionally, you can use the message definition to specify the number of FAO arguments to be included in the message text. Any number of message definitions can follow a .SEVERITY directive (or a .FACILITY directive if no .SEVERITY directive is included).

You can place qualifiers in any order before or after the message text.

You can use the severity level qualifiers either to override the severity level defined in a .SEVERITY directive or to replace .SEVERITY directives in your message source file. Only one Message Definition qualifier can be included per message definition.

EXAMPLE

```
 .TITLE         SAMPLE Error and Warning Messages
 .IDENT         'VERSION 4.00'
 .FACILITY      SAMPLE,1/PREFIX=ABC_   ❶
 .SEVERITY      ERROR

UNRECOG        < Unrecognized keyword !AS>/FAO_COUNT=1 ❷
AMBIG          "Ambiguous keyword"    ❸

 .SEVERITY      WARNING
 .BASE          10
SYNTAX         < Invalid syntax in keyword>     ❹

 .END
```

This message source file contains a .FACILITY directive ❶ and three message definitions ❷ ❸ ❹. The symbol names—UNRECOG, AMBIG, and SYNTAX—specified in the message definitions are combined with a prefix, ABC_, defined in the .FACILITY directive, to form the message symbols, ABC_UNRECOG, ABC_AMBIG, and ABC_SYNTAX.

The message text of the UNRECOG and SYNTAX messages ❷ ❹ is delimited by angle brackets (<>); the message text of the AMBIG message ❸ is delimited by quotation marks ("").

In addition, the first message definition in this example includes the FAO directive !AS (which inserts an ASCII string at the end of the message text) and the corresponding qualifier /FAO_COUNT.

Page Directive

Forces page breaks in the output listing.

FORMAT

.PAGE

PARAMETERS

None.

QUALIFIERS

None.

DESCRIPTION

The .PAGE directive allows you to specify page breaks in the output listing. You can specify only one page break per any one .PAGE directive; however, you can use the .PAGE directive as often as you like.

EXAMPLE

```
 .TITLE         SAMPLE Error and Warning Messages
 .IDENT         'VERSION 4.00'
 .FACILITY      SAMPLE,1/PREFIX=ABC_
 .SEVERITY      ERROR

UNRECOG        < Unrecognized keyword !AS>/FAO_COUNT=1
AMBIG          < Ambiguous keyword>
 .PAGE ❶

 .SEVERITY      WARNING
 .BASE          10
SYNTAX         < Invalid syntax in keyword>

 .END
```

This .PAGE directive ❶ forces a page break in the output listing after the AMBIG message definition.

Severity Directive

Specifies the severity level to be associated with the messages that follow the .SEVERITY directive.

FORMAT

.SEVERITY *level*

PARAMETER

level
Specifies the level of the condition that caused the message. The severity level codes are as follows:

SUCCESS	Produces an S code in a message.
INFORMATIONAL	Produces an I code in a message.
WARNING	Produces a W code in a message.
ERROR	Produces an E code in a message.
SEVERE	Produces an F code in a message.
FATAL	Produces an F code in a message.

The SEVERE parameter is equivalent to the FATAL parameter and they can be used interchangeably.

QUALIFIERS

None.

DESCRIPTION

Following the .FACILITY directive, the message source file generally contains a .SEVERITY directive. If you do not specify the severity on each message definition with one of the Message Definition severity qualifiers, you must include a .SEVERITY directive. If you attempt to define a message without specifying a severity level, an error results.

A new .FACILITY directive cancels the previous severity level in effect.

EXAMPLE

```
 .TITLE         SAMPLE Error and Warning Messages
 .IDENT         'VERSION 4.00'
 .FACILITY      SAMPLE,1/PREFIX=ABC_
 .SEVERITY      ERROR      ❶

UNRECOG        < Unrecognized keyword !AS>/FAO_COUNT=1
AMBIG          < Ambiguous keyword>

 .SEVERITY      WARNING    ❷
 .BASE          10
SYNTAX         < Invalid syntax in keyword>

 .END
```

The two .SEVERITY directives ❶ ❷ included in this message source define the severity levels for three messages. The first two messages have a severity level of E; the third message has a severity level of W.

Title Directive

Specifies the module name and title text that is to appear at the top of each page of the output listing file.

FORMAT

.TITLE *modname [listing-title]*

PARAMETERS

modname
Specifies a character string of up to 31 characters that is to appear in the object module as the module name.

listing-title
Defines the text to be used as the title of the listing. The title begins with the first nonblank character after the module name and continues through the next 28 characters. An exclamation mark (!) within these 28 characters is treated as part of the title and not as a comment delimiter. The listing title has a maximum length of 28 characters and cannot be continued onto another line.

QUALIFIERS

None.

EXAMPLE

```
 .TITLE       SAMPLE Error and Warning Messages❶
 .IDENT       'VERSION 4.00'
 .FACILITY    SAMPLE,1/PREFIX=ABC_
 .SEVERITY    ERROR

UNRECOG      < Unrecognized keyword !AS>/FAO_COUNT=1
AMBIG        < Ambiguous keyword>

 .SEVERITY    WARNING
 .BASE        10
SYNTAX       < Invalid syntax in keyword>

 .END
```

The module name ❶ of the object module produced by this file is SAMPLE, and the title of the output listing ❷ is defined as "Error and Warning Messages."

MESSAGE EXAMPLES

The following examples demonstrate the use of message files and pointer files.

Creating an Executable Image Containing Message Data

The following example illustrates the steps involved in incorporating a message file within an executable image.

The message source file, TESTMSG.MSG, created with a text editor, contains the following lines:

```
.FACILITY        TEST,1 /PREFIX=MSG_
.SEVERITY        ERROR
SYNTAX           < Syntax error in string '!AS'>/FAO=1
ERRORS           < Errors encountered during processing>
.END
```

You compile the message source file by entering the following command:

```
$ MESSAGE TESTMSG
```

You compile the FORTRAN source file by entering the following command:

```
$ FORTRAN TEST
```

You link the message object module, TESTMSG.OBJ, to the FORTRAN object module, TEST.OBJ, by entering the following command:

```
$ LINK/NOTRACE TEST+TESTMSG
```

You execute the image by entering the following command:

```
$ RUN TEST
```

If an error occurs when you execute the program, the system displays the following messages:

```
%TEST-E-SYNTAX, Syntax error in string ABC
%TEST-E-ERRORS, Errors encountered during processing
```

Creating an Executable Image Containing a Pointer

The following example demonstrates how to create an executable image that contains a pointer to a nonexecutable message file.

You compile the message source, COBOLMSG, by entering the following command:

```
$ MESSAGE/NOSYMBOLS COBOLMSG
```

You link the object module, COBOLMSG.OBJ, to create the nonexecutable message file by entering the following command:

```
$ LINK/SHAREABLE=SYS$MESSAGE:COBOLMF COBOLMSG.OBJ
```

You create the pointer object module, MESPNTR.OBJ, which contains a pointer to the nonexecutable message file, COBOLMF.EXE, by entering the following command:

```
$ MESSAGE/FILE_NAME=COBOLMF /OBJECT=MESPNTR COBOLMSG
```

You link the pointer object module, MESPNTR.OBJ, to the COBOL program object module, COBOLCODE.OBJ, by entering the following command:

```
$ LINK COBOLCODE,MESPNTR
```

You execute the program by entering the following command:

```
$ RUN COBOLCODE
```

The system then displays the messages defined in COBOLMSG.

Index

Reader's Comments

VMS Message Utility Manual
AA–LA63A–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less ______________________________

____

____

What I like best about this manual is ______________________________

____

____

What I like least about this manual is ______________________________

____

____

I found the following errors in this manual:

Page	Description
______	______________________________
______	______________________________
______	______________________________
______	______________________________
______	______________________________

Additional comments or suggestions to improve this manual:

____

____

____

____

I am using **Version** ______ of the software this manual describes.

Name/Title ______________________________ Dept. ____________

Company ______________________________ Date ____________

Mailing Address ______________________________

______________________________ Phone ____________

-- Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01–3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

-- Do Not Tear - Fold Here