VAX/VMS Message Utility Reference Manual

Order Number: AA–Z422A–TE

September 1984

This manual describes the VAX/VMS Message Utility.

Revision/Update Information:	This is a new manual.
Software Version:	VAX/VMS Version 4.0

digital equipment corporation
maynard, massachusetts

September 1984

This document was prepared using an in-house documentation production system. All page composition and make-up was performed by TEX, the typesetting system developed by Donald E. Knuth at Stanford University. TEX is a registered trademark of the American Mathematical Society.

MESSAGE Contents

Preface

Intended Audience

This manual is intended for use by programmers and general users.

Structure of This Document

This document is composed of five major parts.

The Format Section is an overview of the Message Utility and is intended as a quick reference guide. The format summary contains the DCL command that invokes the Message Utility, listing all command qualifiers and parameters. The usage summary describes how to invoke and exit from the Message Utility.

The Description Section explains how to use the Message Utility.

The Qualifier Section describes each DCL command qualifier. Qualifiers appear in alphabetical order.

The Command Section describes each message source file statement and qualifier. Message source file statements appear in alphabetical order.

The Examples Section contains examples of commmon operations that you perform with the Message Utility.

Associated Documents

Information about linking object modules and creating executable, nonexecutable and shareable images can be found in the *VAX/VMS Linker Reference Manual*.

Conventions Used in This Document

Convention	Meaning
RET	A symbol with a one- to three-character abbreviation indicates that you press a key on the terminal, for example, ret .
CTRL/x	The phrase CTRL/x indicates that you must press the key labeled CTRL while you simultaneously press another key, for example, CTRL/C, CTRL/Y, CTRL/O.
$ SHOW TIME 05-JUN-1985 11:55:22	Command examples show all output lines or prompting characters that the system prints or displays in black letters. All user-entered commands are shown in red letters.

Convention	Meaning
$ TYPE MYFILE.DAT . . .	Vertical series of periods, or ellipsis, mean either that not all the data that the system would display in response to the particular command is shown or that not all the data a user would enter is shown.
file-spec,...	Horizontal ellipsis indicates that additional parameters, values, or information can be entered.
[logical-name]	Square brackets indicate that the enclosed item is optional. (Square 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 does not include any significant technical changes.

MESSAGE

The VAX/VMS Message Utility allows you to supplement the VAX/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 Qualifiers	**Defaults**
/[NO]FILE_NAME[=file-spec]	*/NOFILE_NAME*
/[NO]LIST[=file-spec]	*/NOLIST (interactive mode)*
/[NO]LIST[=file-spec]	*/LIST (batch mode)*
/[NO]OBJECT[=file-spec]	*/OBJECT*
/[NO]SYMBOLS	*/SYMBOLS*
/[NO]TEXT	*/TEXT*

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(s).

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

If you specify SYS$INPUT, the message source file(s) 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 explicitly stated.

usage summary

Invoking
The following DCL command invokes the Message Utility:

```
$ MESSAGE
```

Exiting
After compiling the message source file, the Message Utility returns the user to DCL command level.

Directing Output
Does not apply.

Privileges/Restrictions
None.

For details about message statements and directives, qualifiers, and parameters in message source files, see the Source File Statements Section.

DESCRIPTION

Messages are normally displayed to the user as a line of alphanumeric codes and text explaining the condition that caused the message to be issued.

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 also 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 the user should be aware of.

% and ,

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

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

The following is a typical message:

```
%TYPE❶  -W-❷  OPENIN❸  , 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

1 Constructing Messages

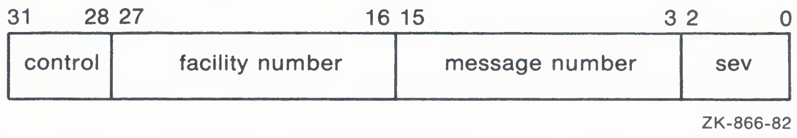
You construct messages by writing a message source file, compiling it using the Message Utility, and linking the resulting object module with your facility object module. When you run your program, the Put Message ($PUTMSG) system service finds the information to use in the message by using a message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

1.1 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

- 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. Following the .TITLE directive, you must specify a .FACILITY 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.

A severity level must be defined 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.

TESTMSG.MSG 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.

1.2 Compiling the Message Source File

Message source files must be compiled into object modules before the messages defined in them can be used. You compile your message source file by issuing the MESSAGE command followed by the file specification of the message source file.

For example, the message source file, TESTMSG.MSG, is compiled by the Message Utility to create an object module, TESTMSG.OBJ, with the command:

```
$ MESSAGE TESTMSG
```

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

1.3 Linking the Message Object Module

After the message file has been compiled, the message object module must be linked with the facility object module (created when the source file was compiled) to produce one executable image file.

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

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

At this point, the program, which contains both the message data and the facility code, can be executed with the command:

```
$ RUN TEST
```

If an error occurs when the program is executed, the following messages will be issued.

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

2 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. Because the message object module is not linked directly with the facility object module, 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 by itself. For example, to create the nonexecutable message file COBOLMF.EXE, you would first create the object module by compiling the message source file, COBOLMSG.MSG, with the following command:

```
$ MESSAGE/NOSYMBOLS COBOLMSG
```

Then, the resulting object module is linked by itself with the 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 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 object module MESPNTR.OBJ, which contains a pointer to the nonexecutable message file COBOLMF.EXE, is created by the command:

```
$ 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 the pointer object module has been created, it can then be linked with the facility object module.

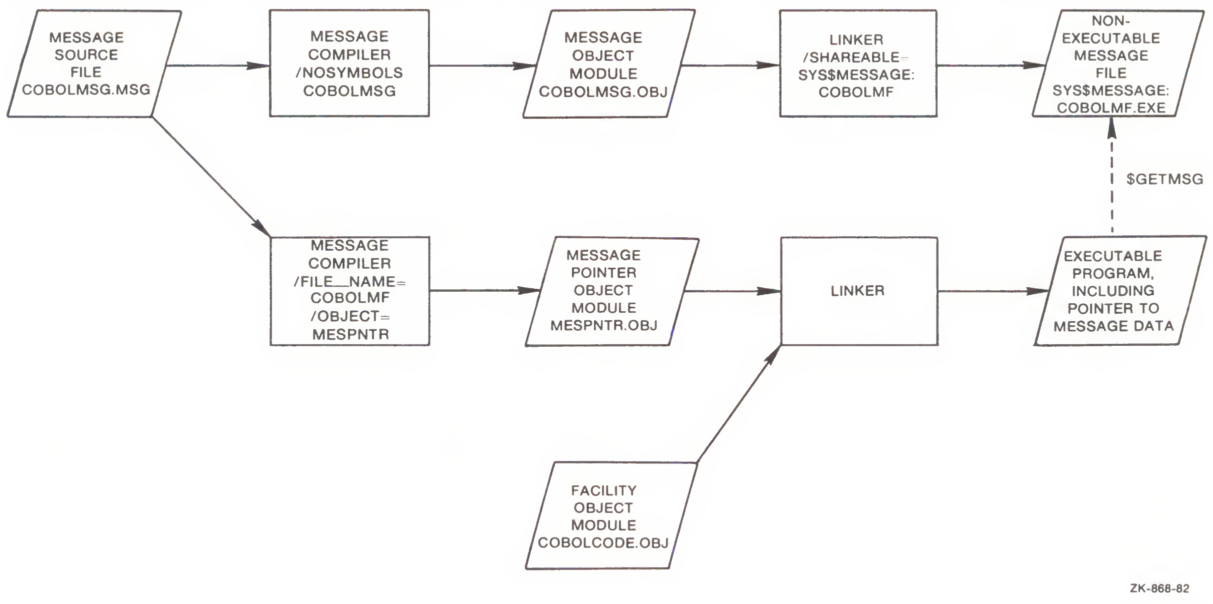
For example, the object module, MESPNTR.OBJ is linked to the COBOL program, COBOLCODE.

```
$ LINK COBOLCODE,MESPNTR
```

When the resulting facility image file is run, the message data is retrieved from the message file, COBOLMF, by the $GETMSG system service.

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

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

3 The SET MESSAGE Command

The SET MESSAGE command allows you to

- 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
```

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 concerning the SET MESSAGE command, see the *VAX/VMS DCL Dictionary*.

4 Error Messages

The *VAX/VMS System Messages and Recovery Procedures Reference Manual* lists the error messages issued by the message compiler and provides explanations and suggested user actions for these messages. Most of the user responses entail changing the message source file and reentering the MESSAGE command.

COMMAND 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 *VAX/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 are mutually exclusive because a message pointer file cannot contain message text. The message text is contained in the nonexecutable message file specified with the /FILE_NAME qualifier.

EXAMPLES

1

```
$ MESSAGE COBOLMSG
```

This MESSAGE command creates the message object, 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 that 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 for listing files is LIS. No wildcard characters are allowed in the file specification.

DESCRIPTION

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. 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 and a file type of LIS.

EXAMPLE

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

This MESSAGE command compiles the message source file COBOLMSG.MSG and creates the output listing MSGOUTPUT.LIS in the 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 with a file type of OBJ.

EXAMPLES

1

```
$ MESSAGE COBOLMSG
```

This MESSAGE command creates the message object, 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

/[NO]SYMBOLS

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 will become the nonexecutable image with the /NOSYMBOLS qualifier as the symbols only have to be 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

/[NO]TEXT

qualifier values *None.*

DESCRIPTION

By default, the message compiler creates an object module that contains text. The /TEXT and /FILE_NAME qualifiers are mutually exclusive since a message pointer file cannot contain message text. The message text is obtained from the nonexecutable message file specified with the /FILE_NAME qualifier.

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

The /NOTEXT qualifier can be used 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.

SOURCE FILE STATEMENTS

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

Message Source File Statements

Message source file statements are imbedded within a message source file. Generally, message source file statements help construct the message code, 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 below.

Any line in the message source file can include a comment delimited by an exclamation point except lines that contain the .TITLE directive. You can 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 to specify the radix of a numeric value:

^X	Hexadecimal
^O	Octal
^D	Decimal

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 also 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 inside to outside.

Base Message Number Directive

Defines the value used in constructing the message code.

FORMAT

.BASE *number*

statement parameter

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

statement 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 another .BASE is encountered or until the end of the messages belonging to the facility.

EXAMPLE

```
.TITLE        SAMPLE Error and Warning Messages
.IDENT        'VERSION 4.00'
.FACILITY     SAMPLE,1/PREFIX=ABC_     ➊
.SEVERITY     ERROR
UNRECOG       < Unrecognized keyword !AS>/FAO_COUNT=1
AMBIG         < Ambiguous keyword>
.SEVERITY     WARNING
.BASE         10        ➋
SYNTAX        < Invalid syntax in keyword>
.END
```

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

End Directive

Terminates the entire list of messages for the facility.

FORMAT **.END**

statement parameters *None.*

statement qualifiers *None.*

DESCRIPTION An End directive terminates the entire list of messages for a facility. In addition to using an end statement to terminate a list of messages for a facility, you may also use another severity directive or a new facility directive.

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>
.END   ❶
```

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

Facility Directive

Specifies the facility to which the messages will apply.

FORMAT

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

statement parameters

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

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

statement 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 /SYSTEM 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.

/SHARED
Inhibits the setting of the facility-specific bit in the message code. The /SHARED qualifier is used only for system service 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. Both the facility name and the facility number are required in a facility directive and can be separated by a comma or by any number of spaces or tabs.

The facility directive creates a global symbol of the form:

```
facnam$_FACILITY
```

This symbol can be used 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 a 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 produced by the Message Utility.

FORMAT .IDENT *string*

statement parameter

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

statement qualifiers

None.

DESCRIPTION The identification directive is in addition to the name you assign to the module with .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 given 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 will be produced by the Message Utility.

Literal Directive

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

FORMAT

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

statement parameters

symbol
Specifies a symbol name.

value
Specifies any valid expression. If the value is omitted, 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.

statement 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        ➊
.LITERAL        NUMSG=(MSG$_LAST@-3)-(MSG$_FIRST@-3)    ! # of messages    ➋
```

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 ➊ defines a symbol that has the value of the last 32-bit message code defined. The second .LITERAL directive ➋ 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,...]*

statement 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 the /IDENTIFICATION qualifier is specified in the message definition.

message-text

Defines the text explaining the condition that caused the message to be issued. 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; these directives are used by the Formatted ASCII Output ($FAO) system service. If you include an FAO directive, you must also use the /FAO_COUNT qualifier.

statement 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 of 0 through 255. The $PUTMSG service uses n to determine how many arguments are to be given to the $FAO service when constructing the final message text. 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 this qualifier is not specified, the name defined in the message definition will be 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, this qualifier must be used 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, this qualifier must be used 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, this qualifier must be used 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, this qualifier must be used to specify the error level.

/SEVERE

Specifies the severe level for a message. This qualifier overrides any .SEVERITY directive in effect. If no .SEVERITY directive is in effect, this qualifier must be used to specify the severe level.

/FATAL

Specifies the fatal level for a message. This qualifier overrides any .SEVERITY directive in effect. If no .SEVERITY directive is in effect, this qualifier must be used to specify the fatal level.

DESCRIPTION

The message definition specifies the message text that will 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.)

Qualifiers can be placed 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 severity 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 will be 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 the angle brackets (<>); the message text of the AMBIG message ❸ is delimited by quotation marks ("").

In addition, the first message definition above 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

statement parameters

None.

statement qualifiers

None.

DESCRIPTION

The .PAGE directive allows you to specify page breaks in the output listing. You can only specify one page break with 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   ➊
.SEVERITY       WARNING
.BASE           10
SYNTAX          < Invalid syntax in keyword>
.END
```

This .PAGE directive ➊ 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.

FORMAT

.SEVERITY *level*

statement parameter

level

Specifies the level of the condition that caused the message.

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.

SEVERE is equivalent to FATAL and they can be used interchangeably; the severity level code for both of these is F.

statement qualifiers

None.

DESCRIPTION

Following the facility directive, the message source file generally contains a severity directive. You must include a severity directive if you do not specify the severity on each message definition with one of the severity qualifiers. If you attempt to define a message without specifying a severity level, an error will result.

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 the severity level W.

Title Directive

Specifies the module name and title text that will appear on the top of each page of the output listing file.

FORMAT

.TITLE *modname [listing-title]*

statement parameters

modname

Specifies a character string of up to 31 characters that will 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 in these 28 characters will be 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.

statement qualifiers

None.

EXAMPLE

```
.TITLE        SAMPLE Error and Warning Messages  ❶
.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 will be "SAMPLE" , and the title of the output listing ❶ will be 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
```

The message source file is compiled with the command

```
$ MESSAGE TESTMSG
```

The FORTRAN source file is compiled with the command

```
$ FORTRAN TEST
```

The message object module, TESTMSG.OBJ, is linked to the FORTRAN object module, TEST.OBJ, with the command

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

You execute the image by issuing the command

```
$ RUN TEST
```

The following messages are issued when the program is executed and an error occurs:

```
%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.

The message source, COBOLMSG, is compiled by itself with the command

```
$ MESSAGE/NOSYMBOLS COBOLMSG
```

The object module COBOLMSG.OBJ, is linked by itself to create the nonexecutable message file with the command

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

The pointer object module, MESPNTR.OBJ, which contains a pointer to the nonexecutable message file, COBOLMF.EXE, is created by the command

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

The pointer object module, MESPNTR.OBJ, is linked to the COBOL program object module, COBOLCODE.OBJ, with the command

```
$ LINK COBOLCODE,MESPNTR
```

The messages defined in COBOLMSG are displayed when the program is executed with the command

```
$ RUN COBOLCODE
```

Index

VAX/VMS
Message Utility
Reference Manual
AA-Z422A-TE

READER'S COMMENTS

Note: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

____

Did you find errors in this manual? If so, specify the error and the page number.

____

Please indicate the type of user/reader that you most nearly represent:

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) ____________________

Name ____________________ Date ________________

Organization ________________________________

Street ________________________________

City ____________________ State ____________ Zip Code or Country ____________

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

SSG PUBLICATIONS ZK1–3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03062–2698

Do Not Tear - Fold Here

Cut Along Dotted Line