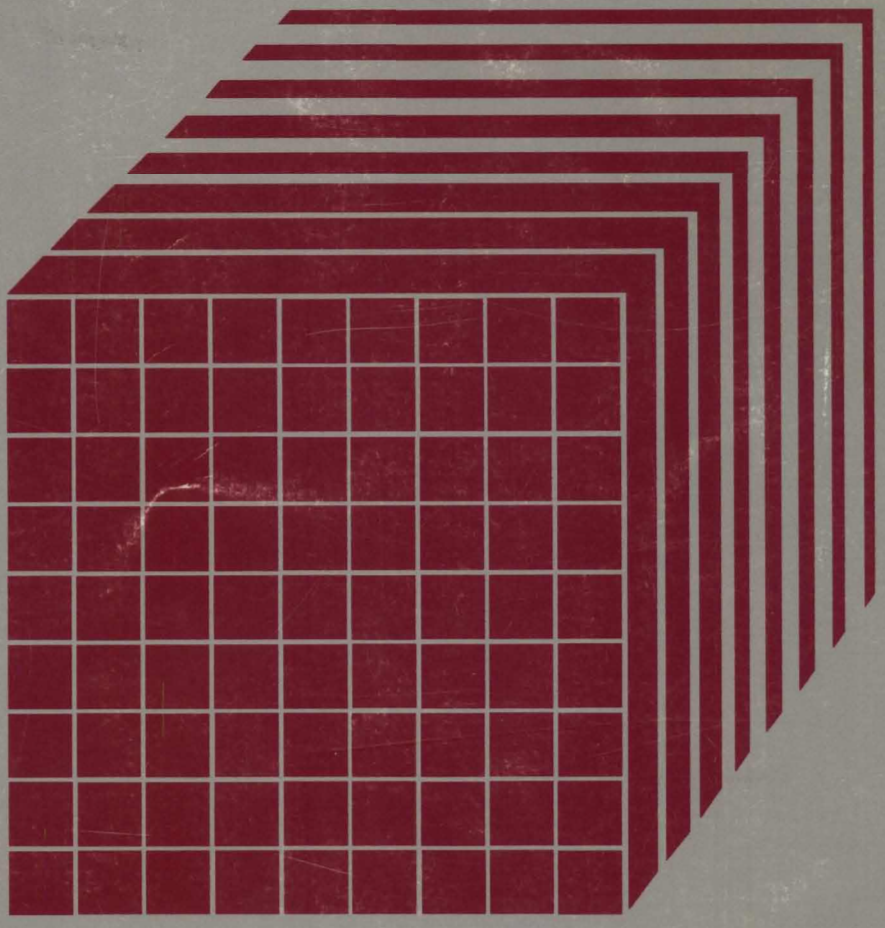
IBM

Virtual Machine/
System Product

CMS Macros and Functions Reference

Release 5

SC24-5284-0



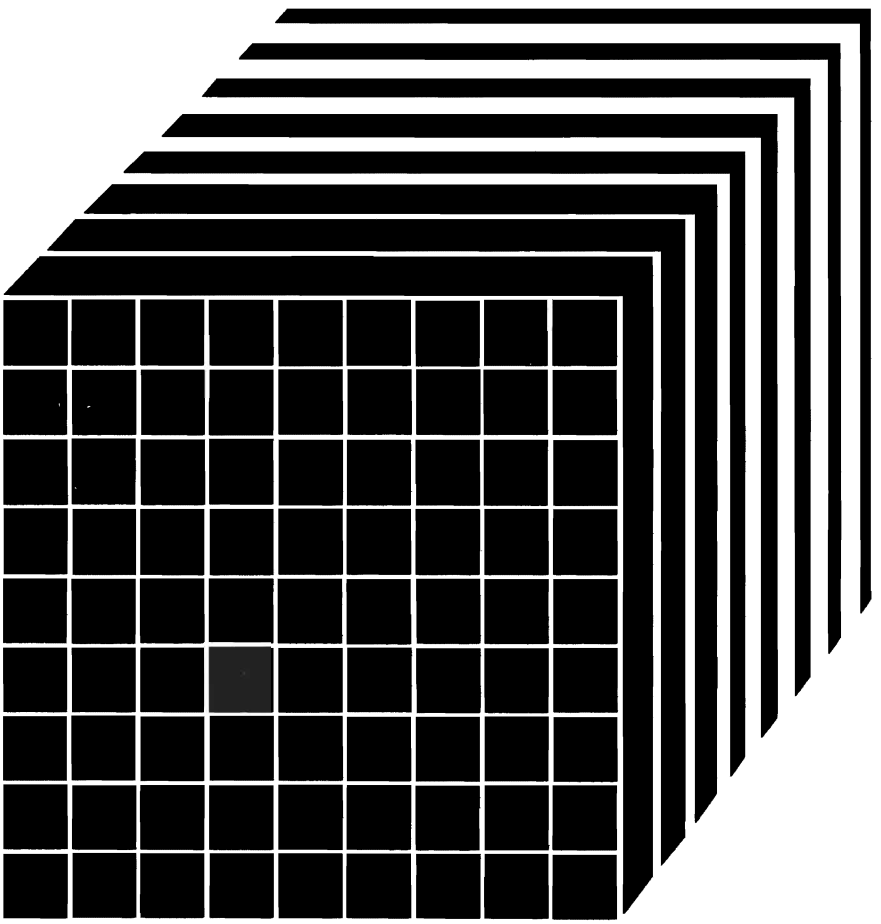
IBM

Virtual Machine/
System Product

CMS Macros and Functions Reference

Release 5

SC24-5284-0



First Edition (December 1986)

This edition, SC24-5284-0, is a revision of the macros and functions information previously contained in *VM/SP CMS Command and Macro Reference*, SC19-6209-3, and applies to Release 5 of Virtual Machine/System Product (VM/SP), program number 5664-167, and to all subsequent versions and modifications until otherwise indicated in new editions or Technical Newsletters. Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370, 30xx, and 4300 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

Summary of Changes

For a list of changes, see "Summary of Changes" on page 189.

Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

Ordering Publications

Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality. Publications are *not* stocked at the address given below.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Information Development, Dept. G60, P.O. Box 6, Endicott, NY, U.S.A. 13760. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Preface

Who Should Read This?

- Application Programmers
- System Programmers
- IBM System Support

What You Should be Able to Do after Reading This Book

- Use CMS macro instructions when you write assembler language programs to execute in the CMS environment.
- Execute CMS functions from assembler language programs.

How to Use This Book

This book has two parts:

- "CMS Macro Instructions" provides detailed reference information about most external CMS assembler language macro instructions.
- "CMS Functions" provides detailed reference information about CMS functions.

Related VM/SP Publications

You can find more information about VM/SP publications in the *VM/SP Library Guide, Glossary, and Master Index.*

For information about certain other CMS assembler language macro instructions, refer to *VM/SP CMS for System Programming* or *VM System Facilities for Programming.*

Contents

Chapter 1. CMS Macro Instructions

This part describes the formats of the CMS assembler language macros. Use these macros when you write assembler language programs to execute in the CMS environment. To assemble a program using any of these macros, you must issue the GLOBAL command specifying MACLIB DMSSP CMSLIB. These macro libraries are normally located on the system disk.

- DMSSP MACLIB contains macros that are new or changed in VM/SP.
- CMSLIB MACLIB contains macros from VM/370.

Note: When assembling programs that use CMS macros, both of these libraries should be identified via the GLOBAL command. DMSSP should precede CMSLIB in the search order.

For functional descriptions and usage examples of the CMS macros, see *VM/SP CMS for System Programming.*

CMS Macro Coding Conventions

Coding conventions for CMS macros are the same as those for all assembler language macros. The macro format descriptions show optional operands in the format:

```
[,operand]
```

indicating that if you are going to use this operand, it must be preceded by a comma (unless it is the first operand coded). If a macro statement overflows to a second line, you must use a continuation character in column 72. No blanks may appear between operands. Incorrect coding of any macro results in assembler errors and MNOTEs.

Where applicable, the end of a macro description contains a list of the possible error conditions that may occur during the execution of the macro, and the associated return codes. These return codes are always placed in register 15. The macros that produce these return codes have ERROR= operands, that allow you to specify the address of an error handling routine, so that you can check for particular errors during macro processing. If an error occurs during macro processing and no error address is provided, execution continues at the next sequential instruction following the macro.

CMS Macro Formats

There are four types of CMS macro formats:

Standard
generates the parameter list inline and is not reentrant. It also generates the code to execute the specified function as part of the macro expansion. CMS macros that result in an SVC 202 use this format.

List
generates a parameter list, but does not generate code to execute the specified function. The parameter list is generated inline and register notation usually cannot be used.

Complex List
generates a parameter list, but does not generate code to execute the specified function. The parameter list is generated in an area that you specify. For example, if the storage area is remote from your program and obtained by GETMAIN, GETVIS, or DMSFREE, the macro can generate reentrant code for your program by using a remote parameter list.

Execute
generates code to execute the specified function. No parameter list is generated by the macro. The parameter list must have previously been created by using the list format of the macro. You may modify parameters in the parameter list by using this format of the macro.

The parameter list passed to the function by the Execute Format must contain a valid combination of parameters with no conflicting options. It must contain all required parameters, without any extraneous parameters from previous macro calls. In many cases, the parameter list must be reinitialized before each new invocation of the Execute Format.

Notes:

1. *Please note that the Standard and Execute Formats of CMS macros alter the contents of registers 1 and 15. The Complex List Format alters the contents of register 1. The contents of register 0 may also be altered when certain macros are invoked. Read the operand descriptions and usage notes carefully; they contain more detail about register usage for the individual macros.*

2. *Not every CMS macro instruction is available in all four formats. Each, however, is available in a Standard Format.*

3. *The following CMS macros provide all four types of formats for the user: ABNEXIT, CMSDEV, CONSOLE, IMMCMD, LINERD, LINEWRT, PARSECMD, and WAITECB.*

ABNEXIT

Use the ABNEXIT macro instruction to set or clear ABEND exit routines. The RESET option clears the condition that indicates control was given to an exit routine.

There are four formats of the ABNEXIT macro instruction:

- Standard
- List (MF=L)
- Complex List (MF=(L,addr[,label]))
- Execute (MF=(E,addr))

Standard Format

The Standard Format of the ABNEXIT macro instruction is:

[label]	ABNEXIT	{SET,EXIT=addr[,UWORD=addr] [,ERROR=addr] CLR,EXIT=addr[,ERROR=addr] RESET[,ERROR=addr]}

where:

label
is an optional statement label.

addr
is an assembler program label or an address stored in a general register. If a register is used, it must be enclosed in parentheses.

SET
establishes an exit routine. This exit routine is added to the list of exit routines and becomes the current exit routine.

CLR
removes the specified exit routine from the list of exit routines. If it was the "current" exit routine, the previous exit routine on the list becomes the "current" exit routine. Exits can be cleared independently of their position in the list.

RESET
clears the condition that indicates control has been given to an exit routine. The RESET option can only be specified from within an exit routine.

EXIT=

You can add or delete an exit routine from the list of exits. The value specified may be a label or a general register. The general register must be specified between parentheses.

label
is an assembler program label that marks the address of the exit routine.

(reg)
is a general register. Its value is the address of the exit routine.

UWORD=

is an optional fullword that can be specified for any purpose you desire. When the exit routine gains control, this fullword is available to the exit in the DMSABW CSECT of DMSNUC.

label
is an assembler program label that is the address stored as the UWORD.

(reg)
is a general register. Its contents are stored as the UWORD.

ERROR=

indicates the address of an error routine to be given control if an error occurs. If ERROR= is not coded and an error occurs, control returns to the next sequential instruction (NSI) in the calling program, as it does if no error occurs.

label
is an assembler program label that is the address of the error routine.

(reg)
is a general register. Its value is the address of the error routine.

List Format

The List Format (MF=L) of the ABNEXIT macro instruction is:

[label]	ABNEXIT MF=L	[[,EXIT=label] [,UWORD=label] ,SET [,EXIT=label] [,UWORD=label] ,CLR [,EXIT=label] ,RESET]

The parameters have the same meaning as in the Standard Format except for the following:

MF=L
indicates that the parameter list is created in-line. No executable code is generated. Register notation cannot be used for macro parameter addresses.

Note: When you use the MF= parameter, all other parameters are optional. Before the function is executed, a valid combination of parameters must be specified by the List and Execute Formats of the macro.

Complex List Format

The Complex List Format (MF=(L,addr[,label])) of the ABNEXIT macro instruction is:

[label]	ABNEXIT MF=(L,addr[,label])	[,EXIT=addr][,UWORD=addr] ,SET[,EXIT=addr][,UWORD=addr] ,CLR[,EXIT=addr] ,RESET

The parameters have the same meaning as in the standard format except for the following:

MF=(L,addr[,label])
indicates that the parameter list is created in the area specified by addr. The address may be a label or may be specified in a register. It represents an area within your program or an area of free storage obtained by a system service. You can determine the size of the parameter list by coding the label operand. The macro expansion equates label to the size of the parameter list. This format of the macro produces executable code to move the data into the parameter list specified by addr. It does *not* generate the instruction to invoke the function. If you use this version of the List Format, you must execute it prior to any related invocation of the Execute Format.

Note: When you use the MF= parameter, all other parameters are optional. Before the function is executed, a valid combination of parameters must be specified by the List and Execute Formats of the macro.

Execute Format

The Execute Format (MF=(E,addr)) of the ABNEXIT macro instruction is:

[label]	ABNEXIT MF=(E,addr)	[,EXIT=addr][,UWORD=addr][,ERROR=addr] ,SET[,EXIT=addr][,UWORD=addr][,ERROR=addr] ,CLR[,EXIT=addr][,ERROR=addr] ,RESET[,ERROR=addr]

The parameters have the same meaning as in the Standard Format except for the following:

MF=(E,addr)
indicates that instructions are generated to execute the ABNEXIT function.

addr
is a label or an address stored in a register that represents the location of the parameter list. Information in the parameter list may be changed by specifying the appropriate operands on the macro.

Note: When you use the MF= parameter, all other parameters are optional. Before the function is executed, a valid combination of parameters must be specified by the List and Execute Formats of the macro.

Usage Notes:

1. You are responsible for providing the proper entry and exit linkage for your abend exit routine. When your routine receives control, the register contents are as follows:

Register	Contents
1	Address of the DMSABW CSECT in DMSNUC. You may use the ABWSECT macro to map the fields of DMSABW in your exit routine. Refer to *VM/SP Data Areas and Control Block Logic Volume 2 (CMS)*, for a description of the fields within ABWSECT.
13	Address of an 18-fullword save area (for your use).
14	Return address.
15	Entry point address of your exit routine.

2. At completion of the abend exit routine, the exit can do one of the following:
   - Return to DMSABN via a branch on register 14. DMSABN will call any previous abend exits if they exist. If not, DMSABN will continue with normal CMS abend recovery.
   - Return elsewhere by loading the PSW, or a modified version of the PSW, at time of abend (available in DMSABW). Prior to loading the PSW, the exit routine should issue an ABNEXIT RESET macro.
3. Abend exits cannot be set or cleared from within an exit routine. You can issue the ABNEXIT macro with the RESET option only from within an exit.
4. In addition to register 1, the macro expansion for RESET uses registers 14 and 15. Your program must have a DSECT for NUCON when

RESET is used or when the Execute Format of the macro is used and the function is not specified.

5. If a program check occurs in the exit (ABNEXIT RESET has not been issued), control is given to the previous exit in the list. Note that the information in the abend work area (ABWSECT) will reflect this secondary error that occurred in the exit, and not the original error. If there are no previous exits, CMS abend recovery occurs.

6. There are two ways to disable an exit routine once it is identified to the system by the ABNEXIT SET macro:

   - Issue the ABNEXIT CLR macro. This can be done at any time except within an exit routine.

   - When CMS abend recovery occurs, CMS automatically clears all exit routines known to the system.

   Abend exits are not cleared at CMS end-of-command.

Error Conditions:

If an error occurs, register 15 contains one of the following return codes:

Code	Meaning
8	SVC issued to clear an exit, but no exits exist for the address
12	ABNEXIT SET or CLR was issued from within an exit routine
16	ABNEXIT RESET was issued outside of an exit
104	Not enough storage available to establish the exit

ADDENTRY

Servers use the ADDENTRY macro to place an entry-name on a Communications Module termination notification list. Each entry on this list is notified (gets control) when CMSSERV communications end. The server entry point to be put on the list of servers to receive control may or may not be a nucleus extension.

The format of the ADDENTRY macro instruction is:

[label]	ADDENTRY	entry-name

where:

label
is an optional statement label.

entry-name
is the name of an entry to be notified when the communications module terminates, either normally or abnormally.

The IBM System/370 to IBM Personal Computer Enhanced Connectivity Facilities return code is returned in register 15.

The following assembly message (MNOTE) may be produced during assembler processing of the ADDENTRY macro:

```
DMSMAC021S  ENTRY NAME NOT SPECIFIED IN ADDENTRY MACRO
```

For more information on the ADDENTRY macro and how to use it with Enhanced Connectivity Facilities on VM/SP, see the *VM/SP IBM Programmer's Guide to the Server-Requester Programming Interface for VM/SP*.

APPLMSG

Use the APPLMSG macro in an assembler program to retrieve a message from a message repository. (A message repository contains translated versions of system messages in the specified language.) You can optionally display the message at your terminal. The format of the APPLMSG macro is:

```
[label]   APPLMSG   [,MF={L|E,(addr)}|(E,(reg)]
                    [,CSECT= {*|NAME}]
                    [,APPLID=applid]
                    [,BUFFA={addr|(reg)}]
                    [,COMP={YES|NO}]
                    [,DISP={ERRMSG|NONE|TYPE|CPMSG|EXECOMM|var}]
                    [,HEADER={YES|NO}]
                   [  ,LET={char|(reg)|*}  ]
                   [  ,LETA={addr|(reg)}   ]
                   [  ,NUM={num|(reg)}     ]
                   [  ,NUMA={addr|(reg)}   ]
                   [  ,FMT={num|(reg)}     ]
                   [  ,FMTA={addr|(reg)}   ]
                   [  ,LINE={num|(reg)|*}  ]
                   [  ,LINEA={addr|(reg)}  ]
                    [,SUB=(sublist)]     [,MAXSUBS=num]
                   [  ,TEXT='message-text' ]
                   [  ,TEXTA={addr|(reg)}  ]
                    [TYPCALL={SVC|NONE}]
```

The APPLMSG macro operands are listed in detail below:

MF Operand

Use the MF operand to specify the macro format. Using different macro formats, you can either code parameters directly in the macro call or put them at a place in the program where they can be referenced later. The macro format must be one of the following:

Standard Format
coded without the MF= operand, generates an in-line operand list and invokes the message facility. This is the default format. You can specify a maximum of 20 substitutions with this macro format.

MF=L (List Format)
is used only together with the Execute Format of APPLMSG. MF=L generates a storage area for the parameter list; this storage area later gets filled in when you use the Execute Format.

The size of the parameter list area you want to reserve depends on the number of substitutions to be made. Use the MAXSUBS operand to specify the size of this area. For example, the following would reserve space for a parameter list that may hold up to five substitutions.

```
MF=L,MAXSUBS=5,...
```

MF=(E,addr) or MF=(E,(reg)) (Execute Format)
generates code to fill in the parameter list at the address you specify, and invokes the message facility. For example:

```
MF=(E,label),...
```

CSECT Operand

Use the CSECT operand to override the default CSECT identifier that will go in the message header. The format of the CSECT operand is:

```
CSECT={*|name}
```

By default, the first three characters of the module name are used, if they are different from the application id; if these three characters are the same as the application id, then the next three characters of the module name are used.

APPLID Operand

Use the APPLID operand to specify the name of the application from which the message is issued. The format of the APPLID operand is:

```
APPLID=applid
```

The three-character application id is compared with the application id in the repository information chain to retrieve the message from the proper repository. It is also displayed in the message header. This parameter is required.

BUFFA Operand

Use the BUFFA operand to specify the address of a buffer where the complete message is to be copied. You can specify this buffer address directly or in a register. The format of the BUFFA operand is:

```
BUFFA={addr|(reg)}
```

Use DISP=NONE when you only want to have the message copied to the buffer and not displayed via the message facility.

When the text is copied into the buffer, the length of the message occupies the first byte of the buffer, preceding the text. The message header (for example, DMSxxxnnns) is also part of the copied information, unless you specify HEADER=NO.

Note: The length of the buffer, not including the length byte, must be placed in the first byte of the buffer before the call to APPLMSG is made. This is done to ensure that the message processor does not overwrite any data immediately following the buffer.

COMP Operand

Use the COMP operand to specify whether multiple blanks in the message text are to be removed, including those preceding and following a substitution field. The format of the COMP operand is:

```
COMP={YES|NO}
```

DISP Operand

Use the DISP operand to specify the display format (disposition) of the message. The format for DISP is one of the following:

DISP=ERRMSG
: specifies that the line is to be checked to see if it qualifies for error message editing. This is the default DISP value unless you specify TEXT or HEADER=NO.

  The standard header format of VM/SP error messages is:

  ```
  xxxmmmnnns or xxxmmmnnnns
  ```

  where:

  - xxx is the application id
  - mmm is the CSECT name
  - nnn or nnnn is the message number
  - s is the severity code.

    The following is a list of the most commonly used severity codes:

Code	Message Type
E	Error
I	Information
R	Response
S	Severe
T	Terminal
W	Warning

  The line is displayed according to the CP EMSG setting. If EMSG is set to:

  - ON - the entire message is displayed, header plus text

- OFF - no message is displayed
- TEXT - only the text portion is displayed
- CODE - only the ten- or eleven-character header is displayed

DISP=NONE
specifies that no output occurs. This option is useful with the BUFFA operand.

DISP=TYPE
specifies that the message is to be displayed on the terminal. This would be the same as DISP=ERRMSG with EMSG ON. This is the default if you specify TEXT or HEADER=NO.

Note: If the message text wraps to a second line, a split may occur in mid-word.

DISP=CPMSG
specifies that the message is to be passed to CP to be issued as a CP message.

DISP=EXECOMM
specifies that the message is to be returned to a variable in the calling EXEC. The complete message is copied into the variable 'MESSAGE', with the first line in 'MESSAGE.1', the second in 'MESSAGE.2', etc.. The number of lines in the message is copied into 'MESSAGE.0'. This is only used when the module issuing APPLMSG is called from an EXEC.

DISP=*variable*
specifies that a variable shows the message display format to be used. The variable must be 1 byte long and the low order 3 bits of the byte must be set to the desired disposition as follows:

```
ERRMSG   = 000
TYPE     = 001
NONE     = 010
CPMSG    = 011
EXECOMM  = 100
```

HEADER Operand

Use the HEADER operand to specify whether you want a header created for the message. The format for the HEADER operand is:

```
HEADER={YES|NO}
```

Note:

You may not specify

- HEADER=NO with the DISP=ERRMSG option, or
- HEADER=YES with the TEXT option.

The repository describes how many digits of the message number to display.

LET and LETA Operands

Use the LET or LETA operand to specify a severity letter for the message. The formats of the LET and LETA operands are:

```
LET={char|(reg)|*}

LETA={addr|(reg)}
```

A default severity code letter is already provided in the message repository; you should use this parameter only when you want to override the provided severity. ("*" specifies that you want the default severity.)

NUM and NUMA Operands

Use the NUM or NUMA operand to specify the number of the message you want. The formats of the NUM and NUMA operands are:

```
NUM={num|(reg)}

NUMA={addr|(reg)}
```

The message number is one to four digits and is used to locate its associated message text in the repository.

If NUMA is used, then the message number should be defined as a halfword. This parameter is required with all formats except the List Format.

FMT and FMTA Operands

Use the FMT or FMTA operand to specify the message format number. The format number is a one- or two-digit number that identifies different versions of the same message which have the same message number. The formats of the FMT and FMTA operands are:

```
FMT={num|(reg)}

FMTA={addr|(reg)}
```

The formats are numbered from 01 to 99. A blank format defaults to 01. A format of 00 is not allowed.

If FMTA is used, the message format should be defined as one byte.

LINE and LINEA Operands

Use the LINE or LINEA operand to specify the line number of a message. The line number is a one- or two-digit number which identifies each line of a multi-line message. The formats of the LINE and LINEA operands are:

```
LINE={num|(reg)|*}

LINEA={addr|(reg)}
```

Lines are numbered from 01 to 99. An asterisk (*) indicates that all lines for a certain message number and format are to be retrieved. You may only specify an asterisk with the LINE option (not with LINEA), and the asterisk must be hard-coded (not used in a register). You may not specify an asterisk for a line number if you use the BUFFA option.

If BUFFA is not specified, then the LINE parameter is defaulted to asterisk. If BUFFA is specified, then LINE is defaulted to 01. A line number of 00 is not allowed. Each line may be up to 240 characters long.

If you use LINEA, you must define the line number as one byte.

SUB Operand

Use the SUB operand to specify the type of substitution to be performed on those portions of the message where substitutions are indicated. The format of the SUB operand is:

```
SUB=(type,(value,length))
          or
SUB=(type,value)
```

The information supplied with the SUB parameter must specify the type of data, its address, and the length of the substitution, or a number which is used to retrieve the substitution information from the repository. If you specify a length, you must enclose the value and length in parentheses. Otherwise, do not enclose the value in parentheses.

You can specify both the value and length using register notation. When you specify the length, it is interpreted to be the length of the input field, except when used with the HEX, HEXA, DEC and DECA parameters. For these parameters, the length represents the length of the converted result. Following are the possible values of type.

DICT,number
: indicates that the substitution is a dictionary item. The value parameter specifies the number of the dictionary item in the repository.

DICT,(reg)
: indicates that the substitution is a dictionary item. The number of the dictionary item is the value in the specified register.

  It is recommended that you use only system keywords to specify a dictionary item. For example, PROFILE, NOPROFILE, or XEDIT.

  You cannot specify a length with DICT.

HEX,(reg)
: converts the value in the specified register.

HEX,expression
converts the given expression. You can specify a length with type HEX; the default is 8 hex digits (4 bytes).

HEXA,address
converts the fullword at the specified address.

HEXA,(reg)
converts the fullword at the address indicated in the specified register.

You may specify a length with type HEXA; the default is 8 hex digits (4 bytes).

For HEX and HEXA, the length specified indicates the number of digits of the converted fullword to be displayed. The word is truncated from the left.

HEX4A,address
converts the data at the specified address.

The value you specify is converted to graphic hexadecimal format and substituted in the message text. Leading zeros are not suppressed.

For HEX4A, a blank character is inserted following every four bytes (eight characters of output). The data to be converted does not have to be on a fullword boundary.

The length field is mandatory with type HEX4A. The length you specify indicates the number of digits of the converted data to be displayed. This length does not include the blanks which are inserted following every four bytes. The data is truncated from the right.

HEX4A,(reg)
converts the data at the address indicated in the specified register.

Note: See above for explanation of *HEX4A*.

DEC,(reg)
converts the value in the specified register.

DEC,expression
converts the given expression.

You may specify a length with type DEC; the default is 15 digits (excluding the sign if the number is negative).

DECA,address
converts the fullword at the specified address.

DECA,(reg)
converts the fullword at the address specified in the indicated register.

The value you specify is converted to graphic decimal format and

substituted in the message text. Leading zeros are suppressed. If the number is negative, a leading minus sign is inserted.

You can specify a length with type DECA; the default is 15 digits (excluding the sign if the number is negative).

For DEC and DECA, the length specified indicates the number of digits of the converted fullword to be displayed, excluding the minus sign. The word is truncated from the left.

CHARA,address
substitutes the character data at the specified address into the message text.

CHARA,(reg)
substitutes the character data at the address indicated in the specified register into the message text.

The length field is mandatory with type CHARA.

CHAR8A,address
substitutes the character data at the specified address into the message text, and inserts a blank character following each eight characters of output.

CHAR8A,(reg)
substitutes the character data at the address indicated in the specified register and inserts a blank character following each eight bytes of output

The length field is mandatory with type CHAR8A. This length indicates the number of actual characters to be displayed, not including the blanks which are inserted after each 8 characters.

MAXSUBS Operand

Use the MAXSUBS operand to reserve program storage to build the parameter list. The format of the MAXSUBS operand is:

```
MAXSUBS=num
```

The number you specify is the maximum number of substitutions, and that determines the size of the area saved. It is used only with the MF=L macro form.

If you specify MAXSUBS and a SUB list, APPLMSG will take the maximum number of substitutions. That is, if you specify both `MAXSUBS=1` and `SUB=(TYPE,(VALUE,LENGTH),TYPE,(VALUE,LENGTH))`, then 2 is the value used. The number of substitutions is multiplied by the amount of space required for each substitution and added to the storage required for the remainder of the parameter list.

The maximum number of substitutions is 20.

TEXT and TEXTA Operands

Use the TEXT or TEXTA operand to directly specify the message text to be used, instead of using a repository. The formats of the TEXT and TEXTA operands are:

```
TEXT='text'

TEXTA={addr|(reg)}
```

If you specify TEXTA, the first byte at the address specified must contain the length of the message text. For example:

```
APPLMSG TEXTA=MESSAGE
          .
          .
          .
MESSAGE  DC    X'16'
         DC    CL22'THIS IS A LINE OF TEXT'
```

The substitution character defaults to '&' if you specify TEXT or TEXTA. A header is not created for the message, so you may not specify the DISP=ERRMSG or HEADER=YES options with TEXT or TEXTA.

NUM, FMT, LINE and LET may not be specified with TEXT or TEXTA. When TEXT or TEXTA is specified, DISP defaults to TYPE.

Note: If you code TEXT or TEXTA to display a message, that message will always appear in the same language, even if your current language changes.

TYPCALL Operand

Use the TYPCALL operand to specify the type of call you want to generate. The format of the TYPCALL operand is:

```
TYPCALL={SVC|NONE}
```

Macro Processing: The processing that takes place in the macro depends on the value of the MF parameter and the TYPCALL parameter.

Standard Format (MF= operand not specified): The macro generates a series of assembler statements which declare the parameters in-line for use by the message processor module (DMSMGM). If you use substitutions whose values and lengths do not use registers or are not hard-coded, non-reentrant code will be generated for this macro format. The macro then generates a call to DMSMGM module depending on the value of the TYPCALL parameter as follows:

TYPCALL=SVC: the macro generates an SVC 202 to call the DMSMGM module

TYPCALL=NONE: no call to the DMSMGM module is generated

MF=L: The macro generates a 'DS' assembler statement to reserve an area of storage for later use by APPLMSG. The size of the area of storage reserved depends on the value of the MAXSUBS parameter. No call to DMSMGM is generated, and no parameter information is set up.

MF=E: The macro generates a series of assembler statements which build a 'record' in the specified buffer area. This record contains the parameters for use by DMSMGM module. The address of the buffer area is specified as follows:

MF=(E,addr) - the buffer area is at the address specified as addr

MF=(E,(reg)) - the buffer area is at the address specified in the register reg

The macro then generates a call to the DMSMGM module depending on the value of the TYPCALL parameter as described for the Standard Format.

Usage Notes:

1. To learn how to make your own message repository refer to "Getting Languages on Your System in VM/SP" in *VM System Facilities for Programming*.

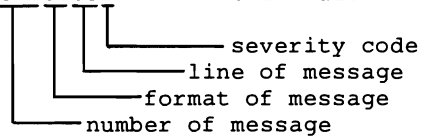
2. APPLMSG contains all the functions of the LINEDIT macro, but it allows you to specify just a message number rather than coding the entire message text. This allows for more flexibility, since a different repository can be in storage and the same message would come up, only it would be in a different language.

3. You should have a copy of the message repository you want to access so that you can see the message numbers, formats, lines, and substitution slots.

Examples:

A message repository may contain the following messages:

```
.
.
08750101E Attempt to divide by &1 is invalid
08750201E Attempt to &2 by &1 is invalid
08760101E Error &1.  rc = &3.
08770101E This is a multi-line message.  NOCOMP must be specified in
08770102E order to keep the return codes lined up on the next line.
08770103E       RC 1 = &1.            RC 2 = &2.
```



A message repository may contain these dictionary items:

```
.
.
90250101  DIVIDE
90260101  reading from &2
90270101  tape
.
.
```

And here is a piece of assembler code that displays error messages when it attempts to divide by zero:

```
SAMP  CSECT
      ENTRY T
*
* TRYING SOME APPLMSG MACRO CALLS
*
T     DS   0H
      LR   10,15
      USING *,10
* SET UP THE REGISTERS FOR THE DIVIDE
      L     3,=F'0'
      L     4,=F'10'
      L     5,=F'0'
*
      CR    3,5              COMPARE REGISTER 5 TO 0
      BE    ERR0             IF REG 5 IS 0, ISSUE AN ERROR MESSAGE
      DR    4,5              OTHERWISE, DO THE DIVIDE
      B     DONE
*
    ------ issue error message; see cases below ------
```

Case 1: This call uses the 'TEXT' parameter to print the message directly, without using the repository:

```
ERR0  APPLMSG APPLID=CMS,COMP=YES,SUB=(DEC,((5),1)),
               DISP=TYPE,TYPCALL=SVC,
               TEXT='ATTEMPT TO DIVIDE BY &1 IS INVALID'
```

Case 2: This call accesses the repository to print CMS message 875, format 1. The parameter list for APPLMSG is set up in-line. (The substitution is a one-digit decimal number in register 5.)

```
APPLMSG  NUM=875,FMT=1,
          APPLID=CMS,COMP=YES,SUB=(DEC,((5),1)),
          DISP=TYPE,TYPCALL=SVC
```

Case 3: This call accesses the repository to print the message. The parameter list for APPLMSG is set up at ERR1. (Again, the substitution is a one-digit decimal number in register 5.)

```
APPLMSG  MF=(E,ERR1),NUM=875,FMT=1,
           APPLID=CMS,COMP=YES,SUB=(DEC,((5),1)),
           DISP=TYPE,TYPCALL=SVC
```

Case 4: This call reserves enough storage for two substitutions.

```
ERR1  APPLMSG MF=L,MAXSUBS=2
```

Case 5: This call uses a dictionary item for the second substitution in the message. The parameter list is set up at ERR1, the location reserved in "Case 4."

```
APPLMSG   MF=(E,ERR1),NUM=875,FMT=02,
           APPLID=CMS,COMP=YES,SUB=(DEC,((5),1),DICT,9025),
           DISP=TYPE,TYPCALL=SVC
```

Note: In this case, the dictionary item is a system keyword. (DICT = DIVIDE)

Messages:

DMSMGM813E *repos* repository not found, message *nnnn* cannot be retrieved RC = 16
DMSMGM814E Message number *nnnn*, format *nn*, line *nn*, was not found; it was called from *routine* in application *applid* RC = 12
DMSMGM815E Invalid double-byte character string *text* replaced by '**' RC = 8

Return Codes:

If an error occurs, register 15 contains one of the following return codes:

Code	Meaning
4	A message was produced, but the text was truncated either because: (1) the user buffer to contain the message text is too short, or (2) the final message text with substitutions is longer than 240 characters. Execution continues.
40	An invalid DISP value was received; processing continues.
104	EXECOMM failed; processing continues.

CMSDEV

Use the CMSDEV macro instruction to obtain the characteristics of a virtual device. The results are returned to a specified storage area.

There are four formats of the CMSDEV macro instruction:

- Standard
- List (MF=L)
- Complex List (MF=(L,addr[,label]))
- Execute (MF=(E,addr))

Standard Format

The Standard Format of the CMSDEV macro instruction is:

[label]	CMSDEV	device,area[,ERROR=erraddr]

where:

label
is an optional statement label.

device
specifies the virtual device whose characteristics are to be obtained. It may be one of the following:

CONS
is a virtual console with an unknown address.

PRT
is the virtual printer.

RDR
is the virtual reader.

PUN
is the virtual punch.

TAP*n*
is a tape device attached to your virtual machine. Valid values for *n* are X'0' to X'F'.

vaddr
is a hexadecimal address of a virtual device that is attached to your virtual machine.

(reg)
is a register (2-12) containing the device address in the low-order two bytes.

area
is the name of a 12-byte storage area to contain the device information. It may be one of the following:

label
an assembler program label that is the address of the storage area.

(reg)
a specified register (2-12) containing the storage area.

ERROR=*erraddr*
indicates the address of an error routine to be given control if the specified device is not attached to your virtual machine (return code 2). If ERROR= is not coded and an error occurs, control returns to the next sequential instruction in the calling program, as it does if no error occurs.

label
is an assembler program label that is the address of the error routine.

(reg)
is a general register (2-12). Its value is the address of the error routine.

List Format

The List Format (MF=L) of the CMSDEV macro is:

[label]	CMSDEV MF=L	[,device] [,area]

The parameters have the same meaning as in the Standard Format except for the following:

MF=L
indicates that the parameter list is created in-line. No executable code is generated. Register notation cannot be used for macro parameter addresses.

Note: When you use the MF=L parameter, all other parameters are optional. Before the function is executed, a valid combination of parameters must be specified by the List and Execute Formats of the macro.

Complex List Format

The Complex List Format (MF=(L,addr[,label])) of the CMSDEV macro is:

[label]	CMSDEV MF=(L,addr[,label])	[,device] [,area]

The parameters have the same meaning as in the standard format except for the following:

MF=(L,addr[,label])
indicates that the parameter list is created in the area specified by addr. The address may be a label or may be specified in a register (2-12). It represents an area within your program or an area of free storage obtained by a system service. You can determine the size of the parameter list by coding the label operand. The macro expansion equates label to the size of the parameter list. This format of the macro produces executable code to move the data into the parameter list specified by addr. It does *not* generate the instruction to invoke the function. If you use this version of the list format, you must execute it prior to any related invocation of the Execute Format.

Note: When you use the MF=(L,addr[,label]) parameter, all other parameters are optional. Before the function is executed, a valid combination of parameters must be specified by the List and Execute formats of the macro.

Execute Format

The Execute Format (MF=(E,addr)) of the CMSDEV macro is:

[label]	CMSDEV MF=(E,addr)	[,device] [,area] [,ERROR=erraddr]

The parameters have the same meaning as in the Standard Format except for the following:

MF=(E,addr)
indicates that instructions are generated to execute the CMSDEV function.

addr
is a label or an address stored in a register (2-12) that represents the location of the parameter list. Information in the parameter list may be changed by specifying the appropriate operands on the macro.

Note: When you use the MF=(E,addr) parameter, all other parameters are optional. Before the function is executed, a valid combination of parameters must be specified by the List and Execute Formats of the macro.

When the CMSDEV macro completes, the defined 12-byte storage area contains the device characteristics.

If the virtual device exists, the first four bytes contain:

Bytes	Virtual Device Information
0	Type class
1	Type
2	Status
3	Flags

If the virtual device is associated with a local real device, bytes four through seven contain:

Bytes	Local Real Device Information
4	Type class
5	Type
6	Model number
7	Current device line length for a virtual console, or the device feature code for other devices.

If the virtual device is associated with a remote real device, bytes four through seven contain:

Bytes	Remote Real Device Information
4	Type class
5	Type for a remote 3270 console
6	Model number for a remote 3270 console
7	Current device line length for a remote virtual console.

If the virtual device is a local virtual console or a remote 3270 virtual console with an unknown address (*device* specified as **CONS**), bytes eight through eleven contain:

Byte	Information
8	The terminal code bits defining the type of virtual console and the translate table the console is using.
9	Reserved
10-11	Virtual device address

For virtual devices other than **CONS**, bytes eight through eleven contain:

Bytes	Information
8	Reserved
9	Reserved
10-11	Virtual device address

Usage Notes:

1. Use the CMSDEV macro when printing with the PRINTL macro to obtain the device characteristics of the virtual printer.

Return Codes:

When the CMSDEV macro completes, register 15 contains one of the following return codes:

Code	Meaning
0	The virtual device is attached and a real device is associated with it.
1	The virtual device is attached and a real device is *not* associated with it. This is normal for spooled devices.
2	The virtual device is not attached or an invalid device address was specified.

COMPSWT

Use the COMPSWT macro instruction to turn the compiler switch (COMPSWT) flag on or off. The COMPSWT flag is in the OSSFLAGS byte of the nucleus constant area (NUCON).

The format of the COMPSWT macro instruction is:

[label]	COMPSWT	{ON / OFF}

where:

label
is an optional statement label.

ON
turns the COMPSWT flag on. When this flag is on, any program called by a LINK, LOAD, XCTL, or ATTACH macro instruction must be a nonrelocatable module in a file with a filetype of MODULE; it is loaded via the CMS LOADMOD command.

OFF
turns the COMPSWT flag off. When this flag is off, any program called by a LINK, LOAD, XCTL, or ATTACH macro instruction must be a relocatable object module residing in a file with a filetype of TEXT, LOADLIB, or TXTLIB; it is loaded via the CMS INCLUDE command.

CONSOLE

Use the CONSOLE macro instruction to access CMS full-screen console services. The CONSOLE macro performs 3270 I/O operations, including building the Channel Command Word (CCW), issuing the DIAGNOSE code X'58' or SIO instruction, waiting for the I/O to complete and checking any error status from the device. Applications must construct a valid 3270 data stream to write to the screen and a 3270 data stream will be returned when a CONSOLE READ is performed.

The CONSOLE macro allows programs to open 'paths' to a display device. The CONSOLE macro coordinates the use of the screen by indicating to an application doing a write that another 'path' has updated the screen last and that the screen must be reformatted. In this manner, full-screen applications will not have to re-write the entire screen every time a write is done.

The CONSOLE macro provides the following functions:

- OPEN/CLOSE - Opening and Closing a specific path to the console
- READ/WRITE/EXCP - Reading and Writing buffers that have 3270 data streams built by the application. The CMS console routine issues the DIAGNOSE code X'58' for the virtual console or a Start I/O (SIO) for dialed devices, tests conditions after I/O, and gives indication back to the application concerning the result of the I/O operation. The CONSOLE function will build the CCW for Read and Write requests, but the application must supply its own CCW when using the EXCP function.
- WAIT - Wait for an interrupt, such as an I/O interrupt from the console device
- QUERY - Get information about the device attributes (from DIAGNOSE code X'24' and X'8C') or about a specific path and its associated device, if the path is opened. The user should provide a buffer to contain this information, which can be mapped by the CQYSECT mapping macro. For more information about the CQYSECT macro, refer to *VM/SP Data Areas and Control Block Logic Volume 2 (CMS).*

The four formats of the CONSOLE macro instruction are:

- Standard
- List (MF=L)
- Complex List (MF=(L,addr[,label]))
- Execute (MF=(E,addr))

Standard Format

The Standard Format for defining a path to a device is:

```
[label]   CONSOLE   OPEN,PATH={'name'          }
                              {(addr[,length])}
                     [,DEVICE=dev]
                     [,EXIT=addr [,UWORD=addr]]
                     [,BUFFER=(addr[,length])]
                     [,ERROR=addr]
```

where:

label
is an optional statement label.

name
is a literal string enclosed in quotes.

addr
is an assembler label or a register (2-12) enclosed in parentheses.

length
specifies the length of the addr. It may be one of the following:

(reg)
is a register containing the length.

n
is an absolute expression whose value is the length in bytes.

dev
specifies the virtual device whose characteristics are to be obtained. It may be one of the following:

vaddr
is a hexadecimal address of a virtual device.

(reg)
is a register (2-12) enclosed in parentheses. If you use a register, the device number is contained in the low-order half and zeros in the remaining bytes.

PATH
specifies the unique name assigned to a path being opened. If you use register notation, you must specify the length of the path name. If you use a label, you can also specify the length. If length is not specified, the length associated with the label is used. The maximum length for a path name is 16 characters (or 16 bytes).

DEVICE
specifies the virtual device number of the console or dialed device associated with the path. You can specify decimal -1 or hex constant FFFFFFFF for the virtual console. If the DEVICE parameter is not explicitly specified, the virtual console is assumed, except in the Complex List and Execute Formats.

EXIT
specifies the address of a routine that gets control in the event of an unsolicited interrupt. Control is given to the exit routine of the path that did the last I/O. If no previous I/O was done, control is given to the exit routine of the path that was last opened. The exit routine receives control as an extension of CMS I/O interrupt handling. The Program Status Word (PSW) is set up with a system storage key and is disabled for interrupts. Register 0 contains the user word. (See UWORD parameter description.)

The exit routine should be prepared to handle all interrupts it receives. You are responsible for establishing proper entry and exit linkage for your routine. When your exit routine receives control, the significant registers contain:

```
Registers Contents
0         UWORD (user word)
13        Address of 72-byte save area
14        Return address
15        Entry point address
```

Your routine must return control to the address specified in register 14 upon entry.

To obtain information about the CSW at the time of interrupt, the interrupting device address, or other information about the path and associated device, the exit should issue the CONSOLE QUERY function, specifying PATH and providing a buffer. Use the CQYSECT macro to map the information moved into the buffer. (See BUFFER parameter description.)

Note: When using the EXIT parameter, you should not have an HNDINT routine defined for the same device associated with this EXIT. The use of CONSOLE macro with the use of HNDINT should be mutually exclusive.

UWORD
specifies an optional fullword parameter that is passed to the exit routine. It can contain any value you wish. When the exit routine gains control, register 0 contains the UWORD parameter (see EXIT parameter description). If you use a label, the address of the label is passed to the routine. If you use a register, the content of the register is passed to the routine.

BUFFER
specifies where information about the opened device is returned to the application. If you use register notation, you must specify the length of the buffer. If you use a label, the length can also be specified. If length is not specified, the length associated with the label is used.

The data returned in the buffer by the OPEN function is mapped by the CQYSECT macro. For more information about the CQYSECT macro, refer to *VM/SP Data Areas and Control Block Logic Volume 2 (CMS)*. If the length of the buffer is less than the length of the data, the data is truncated. You can obtain this information from the CQYSECT length equates. Register 0 contains the length of the data moved into the buffer by the OPEN function.

ERROR
specifies the address of an instruction where execution resumes if an error occurs in processing the CONSOLE request. If this parameter is omitted, execution resumes at the next sequential instruction.

The Standard Format for closing a path to a device is:

[label]	CONSOLE	CLOSE,PATH= {'name' / (addr[,length])} [,ERROR=addr]

where:

The parameters are the same as before.

The Standard Format for waiting for an interrupt from a display device is:

[label]	CONSOLE	WAIT,PATH= {'name' / (addr[,length])} [,ERROR=addr]

where:

The parameters are the same as before.

The Standard Format for obtaining information about a specific device, or a specific path and its corresponding device is:

```
[label]   CONSOLE   QUERY {,PATH= {'name'          }}
                          {      {(addr[,length])}}
                          {,DEVICE=dev             }
                          [,BUFFER=(addr[,length])]
                          [,ERROR=addr]
```

Note: The PATH and DEVICE parameters are mutually exclusive. One of these parameters must be specified. If both are specified, PATH is ignored.

where:

The parameters are the same as before except for the following:

DEVICE
specifies the virtual device number of the console or dialed device being queried. You can specify decimal -1 or hex constant FFFFFFFF for the virtual console. Because this function is used to explicitly query a given path or device, the virtual console is *not* assumed if the DEVICE parameter is not specified.

BUFFER
specifies where information about the device and/or path is returned.

Note: If the buffer is large enough, when you specify PATH, both path and device information are returned. If you specify DEVICE, only information about that device is returned.

The data returned in the buffer from the QUERY parameter is mapped by the CQYSECT macro. For more information about the CQYSECT macro, refer to *VM/SP Data Areas and Control Block Logic Volume 2 (CMS)*. If the length of the buffer is less than the length of the data, the data is truncated. You can obtain this information from the CQYSECT length equates. Register 0 contains the length of the data moved into the buffer from the QUERY parameter.

The Standard Format for writing a 3270 data stream is:

```
[label]   CONSOLE   WRITE,PATH={'name'          }
                               {(addr[,length])}
                        [,BUFFER=(addr[,length])]

                        [,OPTIONS= {option                 }
                                   {(CLEAR[,option])       }
                                   {(NOCLEAR[,option])     }

                                   option:
                                     EW
                                     W
                                     EWA
                                     WSF

                        [,ERROR=addr]
```

where:

The parameters are the same as before except for the following:

BUFFER
specifies the address of an area in storage that contains the 3270 orders and data that is written to the display device. The buffer must contain a complete 3270 data stream.

OPTIONS
specifies optional processing for the buffer. You can specify options in any order. For example, (NOCLEAR,W) or (W,NOCLEAR) are both valid.

The options include:

CLEAR
specifies that the physical screen is cleared before the buffer (if there is one) is written. You can specify this option without the BUFFER parameter to simply clear the screen. If you specify both the BUFFER parameter and the CLEAR option, you should also specify EW, EWA, or WSF.

NOCLEAR
specifies that the physical screen is not cleared by the CONSOLE macro. The operating system may require you to clear the screen manually before the buffer is written. If you do not specify CLEAR or NOCLEAR, NOCLEAR is assumed.

EW
specifies the buffer is written with the Erase/Write option. This option reformats the screen by causing a complete erasure of the screen before the write operation is started. ***You must specify BUFFER with this option.***

W
specifies that the buffer is written with an ordinary Write command, overlaying the current contents of the display screen. ***You must specify BUFFER with this option.*** If you do not specify W, EW, EWA, or WSF, W is assumed.

EWA
specifies the buffer is written with the Erase/Write Alternate option to establish the alternate screen mode for the device. ***You must specify BUFFER with this option.***

WSF
specifies the buffer is written with the Write Structured Field option to provide control information to the device. ***You must specify BUFFER with this option.***

The Standard Format for reading from a display device is:

[label]	CONSOLE	READ,PATH= {'name' / (addr[,length])} ,BUFFER=(addr[,length]) [,OPTIONS= (WAIT|NOWAIT[,RDMOD|,RDBUF])] [,ERROR=addr]

where:

The parameters are the same as before except for the following:

BUFFER
specifies the address of an area in storage where the data is returned from a display device. ***You must specify BUFFER for the READ parameter.*** Register 0 contains the length of the data moved into the buffer from the I/O operation.

OPTIONS
specifies optional processing for the buffer. You can specify options in any order. For example, (WAIT,RDMOD) or (RDMOD,WAIT) are both valid.

The options include:

WAIT
specifies that processing of the request is suspended until an I/O interrupt is received from the device after the last write operation is complete. If WAIT or NOWAIT is not specified, then WAIT is the default.

NOWAIT
specifies that the read request is processed immediately.

RDMOD
specifies that the request is processed as Read Modified and transmits only the modified fields from the screen. If RDMOD or RDBUF is not specified, RDMOD is the default.

RDBUF
specifies that the request is processed as Read Buffer and transmits the entire contents of the screen.

The Standard Format for reading or writing by specifying your own Channel Command Word (CCW) is:

[label]	CONSOLE	EXCP ,PATH= {'name' / (addr[,length])} ,CCW=addr [,ERROR=addr]

where:

The parameters are the same as before except for:

CCW
specifies the address of a channel program containing one or more CCWs that indicate the operation(s) being performed. This parameter is required.

Note: If you issue the EXCP request, you are responsible for generating a valid channel program, necessitating knowledge of the virtual machine architecture and the console support implementation. There is no attempt made to validate the channel program or to convert it to a form appropriate to the implementation. ***The EXCP parameter is not recommended for use with a virtual console.***

List Format

The List Format (MF=L) for defining a path to a device is:

[label]	CONSOLE MF=L	[,OPEN] [,PATH= {'name' / (addr[,length])}] [,DEVICE=dev] [,EXIT=addr[,UWORD=addr]] [,BUFFER=(addr[,length])]

where:

All of the parameters have the same meaning as those in the Standard Format except for:

MF=L
specifies that the parameter list is created in-line. No executable code is generated. You cannot use register notation for macro parameter addresses, and the error parameter is not allowed.

Note: When you use the MF=L parameter, all other parameters are optional. Before the function is executed, you must specify a valid combination of parameters in the List and Execute Formats of the macro.

The List Format for closing a path to a device is:

[label]	CONSOLE MF=L	[,CLOSE] [,PATH= {'name' / (addr[,length])}]

where:

MF=L and other parameters are the same as those mentioned before.

The List Format for waiting for an interrupt from a display device is:

[label]	CONSOLE MF=L	[,WAIT] [,PATH= {'name' (addr[,length])}]

where:

MF=L and other parameters are the same as those mentioned before.

The List Format for getting information about a specific device, or a specific path and its corresponding device is:

[label]	CONSOLE MF=L	[,QUERY] [{,PATH= {'name' (addr[,length])} ,DEVICE=dev}] [,BUFFER=(addr[,length])]

where:

MF=L and other parameters are the same as those mentioned before.

The List Format for writing a 3270 data stream is:

[label]	CONSOLE MF=L	[,WRITE] [,PATH= {'name' (addr[,length])}] [,BUFFER=(addr[,length])] [,OPTIONS= {option (CLEAR[,option]) (NOCLEAR[,option])} option: EW W EWA WSF]

where:

MF=L and other parameters are the same as those mentioned before.

The list format for reading from a display device is:

[label]	CONSOLE MF=L	[,READ] [,PATH= {'name' / (addr[,length])}] [,BUFFER=(addr[,length])] [,OPTIONS= (WAIT|NOWAIT[,RDMOD|RDBUF])]

where:

MF=L and other parameters are the same as those mentioned before.

The list format for reading or writing by specifying your own Channel Command Word (CCW) is:

[label]	CONSOLE MF=L	[,EXCP] [,PATH= {'name' / (addr[,length])}] [,CCW=ccw]

where:

MF=L and other parameters are the same as those mentioned before.

Complex List Format

The Complex List Format (MF=(L,addr[,label])) for defining a path to a device is:

[label]	CONSOLE MF=(L,addr[,label])	[,OPEN] [,PATH= {'name' / (addr[,length])}] [,DEVICE=dev] [,EXIT=addr[,UWORD=addr]] [,BUFFER=(addr[,length])]

All of the parameters have the same meaning as those in the Standard Format except for the following:

MF=(L,addr[,label])
specifies that the parameter list is created in the area specified by addr. You can specify the address as an assembler label or a register (2-12) enclosed in parentheses. It represents an area within a program or an area of free storage obtained by a system service. You can determine the size of the parameter list by coding the label operand. The macro expansion equates label to the size of the parameter list. This format of the macro produces executable code that moves the data into the parameter list specified by addr. It does *not* generate the instruction to invoke the function. If you use this version of the List Format, you must execute it prior to any related invocation of the Execute Format. The ERROR parameter is not valid for this format.

DEVICE
specifies the virtual device number of the console or dialed device associated with the path. You can specify decimal -1 or hex constant FFFFFFFF for the virtual console. If the DEVICE parameter is not explicitly specified, the virtual console is ***not*** assumed.

Note: When you use the MF=(L,addr[,label]) parameter, all other parameters are optional. No default parameters are assumed. Each parameter must be specified. Before the function is executed, you must specify a valid combination of parameters in the List and Execute Format of the macro.

The Complex List Format for closing a path to a device is:

[label]	CONSOLE MF=(L,addr[,label])	[,CLOSE] [,PATH= {'name' / (addr[,length])}]

where:

MF=(L,addr[,label]) and other parameters are the same as those mentioned before.

The Complex List Format for waiting for an interrupt from a display device is:

[label]	CONSOLE MF=(L,addr[,label])	[,WAIT] [,PATH= {'name' / (addr[,length])}]

where:

MF=(L,addr[,label]) and other parameters are the same as those mentioned before.

The Complex List Format for obtaining information about a specific device, or a specific path and its corresponding device is:

```
[label]   CONSOLE MF=(L,addr[,label])   [,QUERY]

                                        [ {,PATH= {'name'          } } ]
                                        [ {       {(addr[,length]) } } ]
                                        [ {,DEVICE=dev               } ]

                                        [,BUFFER=(addr[,length])]
```

where:

MF=(L,addr[,label]) and other parameters are the same as those mentioned before.

The Complex List Format for writing a 3270 data stream is:

```
[label]   CONSOLE MF=(L,addr[,label])   [,WRITE]

                                        [,PATH= {'name'          } ]
                                        [       {(addr[,length]) } ]

                                        [,BUFFER=(addr[,length])]

                                        [,OPTIONS= {option          } ]
                                        [          {(CLEAR[,option]) } ]
                                        [          {(NOCLEAR[,option]) } ]
                                        [                              ]
                                        [  option:                     ]
                                        [    EW                        ]
                                        [    W                         ]
                                        [    EWA                       ]
                                        [    WSF                       ]
```

where:

MF=(L,addr[,label]) and other parameters are the same as those mentioned before.

The Complex List Format for reading from a display device is:

[label]	CONSOLE MF=(L,addr[,label])	[,READ] [,PATH= {'name' / (addr[,length])}] [,BUFFER=(addr[,length])] [,OPTIONS= (WAIT|NOWAIT[,RDMOD|RDBUF])]

where:

MF=(L,addr[,label]) and other parameters are the same as those mentioned before.

The Complex List Format for reading or writing by specifying your own Channel Command Word (CCW) is:

[label]	CONSOLE MF=(L,addr[,label])	[,EXCP] [,PATH= {'name' / (addr[,length])}] [,CCW=ccw]

where:

MF=(L,addr[,label]) and other parameters are the same as those mentioned before.

Execute Format

The Execute Format (MF=(E,addr)) for defining a path to a device is:

[label]	CONSOLE MF=(E,addr)	[,OPEN] [,PATH= {'name' / (addr[,length])}] [,DEVICE=dev] [,EXIT=addr[,UWORD=addr]] [,BUFFER=(addr[,length])] [,ERROR=addr]

where:

All of the parameters have the same meaning as those in the Standard Format except for:

MF=(E,addr)
specifies that instructions are to be generated to execute the CONSOLE function. The address specifies the location of the parameter list. You can specify it as an assembler label or a register (2-12) enclosed in parentheses. You can change information in the parameter list by specifying the appropriate operands on the macro.

DEVICE
specifies the virtual device number of the console or dialed device associated with the path. You can specify decimal -1 or hex constant FFFFFFFF for the virtual console. If the DEVICE parameter is not explicitly specified, the virtual console is ***not*** assumed.

Note: When you use the MF=(E,addr) parameter, all other parameters are optional. No default parameters are assumed. You must specify a valid combination of parameters in the Execute Format of the macro before the function is executed.

The Execute Format for closing a path to a device is:

```
[label]   CONSOLE MF=(E,addr)   [,CLOSE]
                                [,PATH= {'name'        }]
                                        {(addr[,length])}
                                [,ERROR=addr]
```

where:

MF=(E,addr) and other parameters are the same as those mentioned before.

The Execute Format for waiting for an interrupt from a display device is:

```
[label]   CONSOLE MF=(E,addr)   [,WAIT]
                                [,PATH= {'name'        }]
                                        {(addr[,length])}
                                [,ERROR=addr]
```

where:

MF=(E,addr) and other parameters are the same as those mentioned before.

The Execute Format for obtaining information about a specific device, or a specific path and its corresponding device is:

```
[label]   CONSOLE MF=(E,addr)   [,QUERY]
                                [{,PATH= {'name'        }}]
                                 {       {(addr[,length])}}
                                 {,DEVICE=dev             }
                                [,BUFFER=(addr[,length])]
                                [,ERROR=addr]
```

where:

MF=(E,addr) and other parameters are the same as those mentioned before.

The Execute Format for writing a 3270 data stream is:

```
[label]   CONSOLE MF=(E,addr)   [,WRITE]
                                [,PATH= {'name'        }]
                                        {(addr[,length])}
                                [,BUFFER=(addr[,length])]
                                [,OPTIONS= {option                }]
                                           {(CLEAR[,option])      }
                                           {(NOCLEAR[,option])    }
                                             option:
                                               EW
                                               W
                                               EWA
                                               WSF
                                [,ERROR=addr]
```

where:

MF=(E,addr) and other parameters are the same as those mentioned before.

The Execute Format for reading from a display device is:

```
[label]   CONSOLE MF=(E,addr)   [,READ]
                                [,PATH= {'name'        }]
                                        {(addr[,length])}
                                [,BUFFER=(addr[,length])]
                                [,OPTIONS= (WAIT|NOWAIT[,RDMOD|RDBUF]) ]
                                [,ERROR=addr]
```

where:

MF=(E,addr) and other parameters are the same as those mentioned before.

The Execute Format for reading or writing by specifying your own Channel Command Word (CCW) is:

[label]	CONSOLE MF=(E,addr)	[,EXCP] [,PATH= {'name' / (addr[,length])}] [,CCW=addr] [,ERROR=addr]

where:

MF=(E,addr) and other parameters are the same as those mentioned before.

Return Codes:

Upon completion of the requested function, Register 15 will contain one of the following return codes:

For Open Function

0 The path is opened. If a buffer is provided, the length of the data stored in the buffer is returned in Register 0.

1 A path has been opened to the virtual device, but no real device is currently connected to that virtual device.

24 The plist is invalid; a path was not specified.

28 The path is already open. If a buffer is provided, the length of the data stored in the buffer is returned in Register 0.

40 The virtual device is invalid or not defined.

88 The virtual device is not supported by the Console Facility for full-screen I/O.

104 Unable to obtain storage to process the request.

For Close Function

0 The path is closed.

3 The requested path has been closed, but other paths to the associated device are still open.

24 The plist is invalid; a path was not specified.

28 Path not found.

For Query Function

0 If querying a path, the path is open. If querying a device, the device is defined, connected to a real device, and supported by the Console Facility. If a buffer is provided, the length of data stored in the buffer is returned in Register 0.

1 The virtual device is defined and supported by the Console Facility, but it is not currently connected to a real device.

24 The plist is invalid; a path or a device must be specified.

28 Path not found.

40 The virtual device is invalid or not defined.

88 The virtual device associated with the path is not supported by the Console Facility for full-screen I/O.

For Read/Write/Excp Functions

0 I/O successful.

1 A path has been opened to the virtual device, but no real device is currently connected to that virtual device.

2 You must issue a CONSOLE QUERY for the device before any more I/O is requested. The device characteristics have changed because a device was disconnected and then reconnected.[1]

24 The plist is invalid; the function name is unknown, a required parameter is missing, or conflicting options were specified.

28 Path not found. This return code occurs if the path was never opened, or if a device receives an I/O error because it was detached after the path was opened. The Console Facility closes all paths associated with the device, and indicates that the path no longer exists.

32 A full-screen read or write was requested, but another application wrote to the screen. For a read request, the screen read may not belong to your application. An Erase/Write must be issued to reformat the screen and return ownership to the current application.

100 An I/O error has occurred. You can obtain the CSW status by issuing a CONSOLE QUERY and specifying a buffer that will contain the information.

---

[1] If another application attempts an OPEN and the device characteristics do not match what is currently in the device table, other paths are notified (by return code 2) of this change the next time they attempt to do I/O using the CONSOLE macro.

For Wait Function

0 The WAIT completed successfully.

2 You must issue a CONSOLE QUERY for the device before any more I/O is requested. The device characteristics have changed because a device was disconnected and then reconnected. (See footnote 1)

24 The plist is invalid; a path was not specified.

28 Path not found.

CPRB

The CPRB macro builds a CPRB DSECT (the default) or builds code to acquire storage for and partially initialize a CPRB control block.

The CPRB control block is built on a doubleword boundary.

The format of the CPRB macro for assembler language is given in the *VM/SP IBM Programmer's Guide to the Server-Requester Programming Interface for VM/SP*.

When coded with DSECT=YES (or without the DSECT parameter), the CPRB macro builds a DSECT that you should use to access CPRB fields. You can use the label CRBCB to address the CPRB with an assembler USING statement.

When coded with DSECT=NO, the CPRB macro builds code to acquire storage for and partially initialize a CPRB control block. The address of the CPRB is returned in register 1. DSECT=NO is required only for issuing SENDREQ.

Except in the case of abend recovery, VM does not automatically release storage allocated by DMSFREE. Therefore, the server should always explicitly release CPRBs that it has acquired by using the DMSFRET macro instruction. The label CRBLEN can be used to specify the length of the CPRB when issuing a DMSFRET macro to free the CPRB storage when the CPRB is no longer required.

The format of the CPRB macro instruction is:

[label]	CPRB	[DSECT=YES|NO]

where:

label
is an optional statement label.

DSECT=YES|NO
YES is the default if the DSECT parameter is not coded. YES means that a DSECT defining the fields in a CPRB will be included. No CPRB control block will be built by the macro call.

NO means a CPRB control block will be built.

The following assembly message (MNOTE) may be produced during assembler processing of the CPRB macro:

```
DMSMAC001E  DSECT OPERAND NOT 'YES' OR 'NO', 'NO' ASSUMED
```

For more information on the CPRB macro and how to use it with Enhanced Connectivity Facilities on VM/SP, see the *VM/SP IBM Programmer's Guide to the Server-Requester Programming Interface for VM/SP*.

CSMRETCD

Servers use this macro to define names for IBM System/370 to IBM Personal Computer Enhanced Connectivity Facilities return codes for VM/SP. The name of each return code value is listed in the *VM/SP IBM Programmer's Guide to the Server-Requester Programming Interface for VM/SP*.

The format for the CSMRETCD macro instruction is:

[label]	CSMRETCD	

where:

label
is an optional statement label.

CSMRETCD does not produce an assembly message (MNOTE).

For more information on the CSMRETCD macro and how to use it with Enhanced Connectivity Facilities on VM/SP, see the *VM/SP IBM Programmer's Guide to the Server-Requester Programming Interface for VM/SP*.

DELENTRY

Servers use the DELENTRY macro to drop entry-names previously placed on the Communications Module termination notification list via the ADDENTRY macro.

The format of the DELENTRY macro instruction is:

[label]	DELENTRY	entry-name

where:

label
is an optional statement label.

entry-name
is the name of an entry to be dropped from the notification list. This name must be one of the entries that was registered via ADDENTRY.

The IBM System/370 to IBM Personal Computer Enhanced Connectivity Facilities return code is returned in register 15.

The following assembly message (MNOTE) may be produced during assembler processing of the DELENTRY macro:

```
DMSMAC031S  ENTRY NAME NOT SPECIFIED IN DELENTRY MACRO
```

For more information on the DELENTRY macro and how to use it with Enhanced Connectivity Facilities on VM/SP, see the *VM/SP IBM Programmer's Guide to the Server-Requester Programming Interface for VM/SP*.

FSCB

Use the FSCB macro instruction to create a file system control block (FSCB) for a CMS input or output disk file.

The format of the FSCB macro instruction is:

[label]	FSCB	['fileid'] [,RECFM=format] [,BUFFER=buffer] [,FORM=E] [,BSIZE=size], [,RECNO=number] [,NOREC=numrec]

where:

label
is an optional statement label.

fileid
specifies the CMS file identifier, which must be enclosed in single quotation marks and separated by blanks ('filename filetype filemode'). If filemode is omitted, A1 is assumed.

RECFM = *format*
indicates whether the records are fixed (F)- or variable (V)-length format. The default is F.

BUFFER = *buffer*
specifies the address of an I/O buffer, from which records are to be read or written.

FORM = **E**
specifies the extended format FSCB is to be generated. This extended format FSCB allows you to specify a value (up to 2^{31}-1) for RECNO and NOREC. If you do not specify FORM = E, the RECNO and NOREC values cannot exceed 65533.

BSIZE = *size*
specifies the number of bytes to be read or written for each read or write request.

RECNO = *number*
specifies the record number of the next record to be accessed, relative to the beginning of the file, record 1. The default is 0, which indicates that records are to be accessed sequentially.

NOREC = *numrec*
specifies the number of records to be read in the next read operation. The default is 1.

Usage Notes:

1. The format of the FSCB macro is as follows:

```
FSCBCOMM  DC   CL8' '     File system command
FSCBFN    DC   CL8' '     Filename
FSCBFT    DC   CL8' '     Filetype
FSCBFM    DC   CL2' '     Filemode
FSCBITNO  DC   H'0'       Relative record number(RECNO)
FSCBBUFF  DC   A'0'       Address of buffer(BUFFER)
FSCBSIZE  DC   F'0'       Number of bytes to read or write(BSIZE)
FSCBFV    DC   CL2'F'     Record format--F or V (RECFM)
FSCBFLG   EQU  FSCBFV+1   Flag byte
FSCBNOIT  DC   H'1'       Number of records to read or write(NOREC)
FSCBNORD  DC   AL4(0)     Number of bytes actually read
FSCBAITN  DC   AL4(0)     Extended FSCB relative record number
FSCBANIT  DC   AL4(1)     Extended FSCB relative number of records
FSCBWPTR  DC   AL4(0)     Extended FSCB relative write pointer
FSCBRPTR  DC   AL4(0)     Extended FSCB relative read pointer
```

2. The options RECFM, BUFFER, BSIZE, RECNO, and NOREC must all be specified as self-defining terms.

3. You can use the same FSCB to reference several different files; you can override the fileid, or any of the options, on the FSOPEN, FSWRITE, or FSREAD macro instructions when you reference a file via its FSCB. However, if the FSOPEN macro instruction is used to ready an existing file, the BSIZE and RECFM fields in the FSCB are reset to reflect actual file characteristics.

4. You can use multiple FSCBs to reference the same file, for example, if you wanted one FSCB for writing and a different FSCB for reading the file. Keep in mind, however, that the file characteristics are inherent to the file and not to the FSCB. If you establish a read or write pointer using the RECNO option in one FSCB, that pointer remains unchanged unless you specify the RECNO option again on the same or any other FSCB for that file.

5. If a fileid is created with a blank filename and/or filetype, return code 28 is given.

FSCBD

Use the FSCBD macro instruction to generate a DSECT for the file system control block (FSCB).

The format of the FSCBD macro instruction is:

[label]	FSCBD	

where:

label
is an optional statement label. The first statement in the FSCBD macro expansion is labeled FSCBD.

Usage Notes:

1. The FSCBD macro instruction expands as follows:

```
          FSCBD
FSCBD     DSECT
FSCBCOMM  DS      CL8         File system command
FSCBFN    DS      CL8         Filename
FSCBFT    DS      CL8         Filetype
FSCBFM    DS      CL2         Filemode
FSCBITNO  DS      H           Relative record number(RECNO)
FSCBBUFF  DS      A           Address of buffer(BUFFER)
FSCBSIZE  DS      F           Number of bytes to read or write(BSIZE)
FSCBFV    DS      CL2         Record format--F or V (RECFM)
FSCBFLG   EQU     FSCBFV+1    Flag byte
FSCBNOIT  DS      H           Number of records to read or write(NOREC)
FSCBNORD  DS      A           Number of bytes actually read
FSCBAITN  DS      F           Extended FSCB relative record number
FSCBANIT  DS      F           Extended FSCB relative number of records
FSCBWPTR  DS      F           Extended FSCB relative write pointer
FSCBRPTR  DS      F           Extended FSCB relative read pointer
```

2. You can use the labels established in the FSCB DSECT to modify the fields in an FSCB for a particular file. An FSCB is created explicitly by the FSCB macro instruction, and implicitly by the FSREAD, FSWRITE, and FSOPEN macro instructions.

3. If you specify FORM=E as the parameter of the FSCB macro instruction, the fields FSCBITNO and FSCBNOIT are no longer used. They are replaced with FSCBAITN and FSCBANIT. The X'20' bit of the FSCBFLG flag is turned on. The fields FSCBWPTR and FSCBRPTR are used by the FSPOINT function. FORM=E plists must be used to manipulate files larger than 65,533 items.

FSCLOSE

Use the FSCLOSE macro instruction to close an open file.

The format of the FSCLOSE macro instruction is:

[label]	FSCLOSE	{fileid [,FSCB=fscb] FSCB=fscb} [,ERROR=erraddr]

where:

label
is an optional statement label.

fileid
specifies the CMS file identifier. It may be:

'fn ft fm'
fileid enclosed in single quotation marks and separated by blanks. If fm is omitted, A1 is assumed.

(reg)
a register other than 0 or 1 containing the address of the fileid (18 characters). When register format is used, the fileid must be exactly 18 characters in length; 8 for the filename, 8 for the filetype, and 2 for the filemode. Shorter names must be filled with blanks.

FSCB=*fscb*
specifies the address of an FSCB. It may be:

label
the label on the FSCB macro instruction.

(reg)
a register containing the address of an FSCB.

ERROR=*erraddr*
specifies the address of an error routine to be given control if an error is found. If ERROR= is not coded and an error occurs, control returns to the next sequential instruction in the calling program, as it does if no error occurs.

Usage Notes:

1. Although CMS routines close files when a command or program completes execution, you must use the FSCLOSE macro instruction when you are executing a program from within an EXEC, or when you are going to read and write records in the same file.

2. If you specify both fileid and FSCB, the fileid is used to fill in the FSCB.

3. Even though an FSCLOSE macro is issued for a file, the directory is not updated on disk as long as there are other files open for output on that disk.

Error Conditions:

If an error occurs, register 15 contains the following error code:

Code	Meaning
6	File is not open or no read or write was issued to file.

FSERASE

Use the FSERASE macro instruction to delete a CMS disk file.

The format of the FSERASE macro instruction is:

[label]	FSERASE	{fileid [,FSCB] / FSCB=fscb} [,ERROR=erraddr]

where:

label
is an optional statement label.

fileid
specifies the CMS file identifier. It may be:

'fn ft fm'
fileid enclosed in single quotation marks and separated by blanks. If fm is omitted, A1 is assumed.

(reg)
a register other than 0 or 1 containing the address of the fileid (18 characters). When register format is used, the fileid must be exactly 18 characters in length; 8 for the filename, 8 for the filetype, and 2 for the filemode. Shorter names must be filled with blanks.

FSCB=*fscb*
specifies the address of an FSCB. It may be:

label
the label of an FSCB macro instruction.

(reg)
a register containing the address of an FSCB.

ERROR=*erraddr*
specifies the address of an error routine to be given control if an error occurs. If ERROR= is not coded and an error occurs, control returns to the next sequential instruction in the calling program, as it does if no error occurs.

Usage Notes:

1. On return from the FSERASE macro, register 1 points to a parameter list. The second doubleword contains the filename; the third doubleword contains the filetype; and the next halfword contains the filemode of the file.

2. If fileid and FSCB= are both coded, the fileid is used to fill in the FSCB.

Error Conditions:

If an error occurs, register 15 contains one of the following error codes:

Code	Meaning
24	Parameter list error
28	File not found
36	Disk not accessed

FSOPEN

Use the FSOPEN macro instruction to ready a file for either input or output.

The format of the FSOPEN macro instruction is:

[label]	FSOPEN	{fileid [,FSCB=fscb] / FSCB=fscb} [,ERROR=erraddr] [,options] [,FORM=E]

where:

label
is an optional statement label.

fileid
specifies the CMS file identifier. It may be:

'fn ft fm'
the fileid enclosed in single quotation marks and separated by blanks. If fm is omitted, A1 is assumed.

(reg)
a register other than 0 or 1 containing the address of the fileid (18 characters). When register format is used, the fileid must be exactly 18 characters in length; 8 for the filename, 8 for the filetype, and 2 for the filemode. Shorter names must be filled with blanks.

FSCB=*fscb*
specifies the address of an FSCB. It may be:

label
the label on an FSCB macro instruction.

(reg)
a register containing the address of an FSCB.

ERROR=*erraddr*
specifies the address of an error routine to be given control if an error is found. If ERROR= is not coded and an error occurs, control returns to the next sequential instruction in the calling program, as it does if no error occurs.

FORM=E
must be specified when the extended format is being used.

Options:

You can specify any of the following FSCB macro options on the FSOPEN macro instruction:

BUFFER = buffer
RECNO = number
BSIZE = size
RECFM = format
NOREC = numrec

These options may be specified either as the actual value (for example, NOREC = 1) or as a register that contains the value (for example, NOREC = 3) where register 3 contains the value 1).

When you use any of these options, the associated field in the FSCB is modified.

Usage Notes:

1. On return from the FSOPEN macro, register 1 points to the FSCB for the file. If no FSCB exists, one is created in the FSOPEN macro expansion. However, if the FSOPEN macro instruction is used to ready an existing file, the BSIZE and RECFM fields are reset to reflect actual file characteristics.

2. If you code both fileid and FSCB =, the fileid is used to fill in the FSCB.

3. You can use the FSOPEN macro instruction to verify the existence of a file to be opened for reading or writing, and you can use FSOPEN to create an FSCB for that file.

Error Conditions:

If an error occurs, register 15 contains one of the following error codes:

Code	**Meaning**
20	Invalid file identifier
28	File does not exist

FSPOINT

Use the FSPOINT macro instruction to reset the write and/or read pointers for a file. The format of the FSPOINT macro instruction is:

[label]	FSPOINT	{fileid [,FSCB=fscb]} [,ERROR=erraddr] {FSCB=fscb} [,WRPNT=wrpnt] [,RDPNT=rdpnt] [,FORM=E]

where:

label
is an optional statement label.

fileid
specifies the CMS file identifier. It may be:

'fn ft fm'
the fileid enclosed in quotation marks and separated by blanks. If fm is omitted, A1 is assumed.

(*reg***)**
a register other than 0 or 1 containing the address of the fileid (18 characters).

FSCB=*fscb*
specifies the address of an FSCB. It may be:

label
the label of an FSCB macro instruction.

(*reg***)**
a register containing the address of an FSCB.

ERROR=*erraddr*
specifies the address of an error routine to be given control if an error is found. If **ERROR=** is not coded and an error occurs, control returns to the next sequential instruction in the calling program, as it does if no error occurs.

WRPNT=*wrpnt*
specifies the new value of the write pointer.

number
any assembler symbol or number.

(reg)
a register containing the binary number.

RDPNT = *rdpnt*
specifies the new value of the read pointer.

number
any assembler symbol or number.

(reg)
a register containing the binary number.

FORM = **E**
must be specified when the extended format FSCB is being used.

Usage Notes:

1. Both write and read pointers may be changed at the same time, and zero indicates no change.

2. Minus one (-1) used for a write pointer indicates that the next item is to be put at the end of the file.

Error Conditions:

If an error occurs, register 15 contains one of the following error codes:

Code	Meaning
1	File not found
2	Parameter list error

FSREAD

Use the FSREAD macro instruction to read a record from a disk file into an I/O buffer.

The format of the FSREAD macro instruction is:

[label]	FSREAD	{fileid [,FSCB=fscb]} [,ERROR=erraddr] [,FORM=E] {FSCB=fscb } [,options]

where:

label
is an optional statement label.

fileid
specifies the CMS file identifier. It may be:

'fn ft fm'
the fileid enclosed in single quotation marks and separated by blanks. If fm is omitted, A1 is assumed.

(reg)
a register other than 0 or 1 containing the address of the fileid (18 characters). When register format is used, the fileid must be exactly 18 characters in length; 8 for the filename, 8 for the filetype, and 2 for the filemode. Shorter names must be filled with blanks.

FSCB=*fscb*
specifies the address of an FSCB. It may be:

label
the label of an FSCB macro instruction.

(reg)
a register containing the address of an FSCB.

ERROR=*erraddr*
specifies the address of an error routine to be given control if an error is found. If ERROR= is not coded and an error occurs, control returns to the next sequential instruction in the calling program, as it does if no error occurs.

FORM=E
must be specified when the extended format FSCB is being used.

Options:

You can specify any of the following FSCB macro options on the FSREAD macro instruction:

BUFFER=buffer
NOREC=numrec
BSIZE=size
RECNO=number

These options may be specified either as the actual value (for example, NOREC=1) or as a register that contains the value (for example, NOREC=(3) where register 3 contains the value 1).

When you use any of these options, the associated field in the FSCB is modified.

Usage Notes:

1. If an FSCB macro instruction has not been coded for a file (and the FSCB= operand is not coded), you must specify the BUFFER= and BSIZE= options to indicate the address of the buffer and its length. When reading variable-length records, a record that is longer than the buffer length is truncated. FSREAD does not clear the buffer when the record length is not the maximum.

2. On return from the FSREAD macro, register 1 points to the FSCB for the file. If no FSCB exists, one is created following the FSREAD macro instruction.

3. If you specify both fileid and FSCB=, the fileid is used to fill in the FSCB.

4. Register 0 contains, after the read operation is complete, the number of bytes actually read. This information is also contained in the FSCBNORD field of the FSCB. Only when zero records are read is the EOF raised on a multiple record read. EOF is not raised when a partial read occurs because fewer records remained than were requested.

5. To read records sequentially, beginning with a particular record number, use the RECNO option to specify the first record to be read. On the next FSREAD macro instruction, use RECNO=0 so that reading continues sequentially, following the first record read.

Error Conditions:

If an error occurs, register 15 contains one of the following error codes:

Code	Meaning
1	File not found
2	Invalid buffer address
3	Permanent I/O error
5	Number of records to be read is less than or equal to zero (or greater than 32,768 for an 800-byte formatted disk)
7	Invalid record format (only checked when the file is first opened for reading)
8	Incorrect length - buffer size too small for item read.
9	File open for output (for an 800-byte formatted disk)
11	Number of records greater than 1 for variable-length file
12	End of file, or record number greater than the number of records in data set
13	Variable-length file has invalid displacement in active file table
14	Invalid character in filename
15	Invalid character in filetype
19	An I/O error occurred on an FBA device. This was indicated by a non-zero condition code from a DIAGNOSE code X'20'. Error detected in module DMSDIO.
25	Insufficient free storage available for file management control areas.
26	Requested item number is negative or item number plus number of items exceeds file system capacity.

FSSTATE

Use the FSSTATE macro instruction to determine whether a particular file exists.

The format of the FSSTATE macro instruction is:

[label]	FSSTATE	{fileid [,FSCB=fscb]} [,ERROR=erraddr] {FSCB=fscb } [,FORM=E]

where:

label
is an optional statement label.

fileid
specifies the CMS file identifier. It may be:

'fn ft fm'
the fileid enclosed in single quotation marks and separated by blanks. If fm is omitted, A1 is assumed.

(*reg***)**
a register other than 0 or 1 containing the address of the fileid (18 characters). When register format is used, the fileid must be exactly 18 characters in length; 8 for the filename, 8 for the filetype, and 2 for the filemode. Shorter names must be filled with blanks.

FSCB=*fscb*
specifies the address of an FSCB. It may be:

label
the label on an FSCB macro instruction.

(*reg***)**
a register containing the address of an FSCB.

ERROR=*erraddr*
specifies the address of an error routine to be given control if an error is found. If ERROR= is not coded and an error occurs, control returns to the next sequential instruction in the calling program, as it does if no error occurs.

FORM=E
must be specified when the extended format FSCB is being used.

Usage Notes:

1. If the specified file exists, register 15 contains a 0 return code.

2. The FSSTATE macro creates a copy of the file status table (FST) in the STATEFST area. Upon return, register 1 points to the input parameter list. The address of the STATEFST is located at X'1C' within the parameter list.

   The file status table contains the following information:

Decimal Displacement	Field Description
0	Filename
8	Filetype
16	Date (mmdd) last written
18	Time (hhmm) last written
20	Write pointer (number of item)
22	Read pointer (number of item)
24	Filemode
26	Number of records in file
28	Disk address of first chain link
30	Record format (F/V)
31	FST Flag Byte
32	Logical record length
36	Number of 800-byte data blocks
38	Year (yy) last written

   For FORM=E, the following are included:

Decimal Displacement	Field Description
40	Alternate file origin pointer
44	Alternate number of data blocks
48	Alternate item count
52	Number of pointer block levels
53	Length of pointer element
54	Alternate date/time (yy mm dd hh mm ss)
60	Reserved

3. The FSSTATE macro disregards the filemode number specified when both the filename and filetype are explicitly specified. When the filename or filetype (or both) are specified as asterisk (*), the filemode number is respected.

Error Conditions:

If an error occurs, register 15 contains one of the following error codes:

Code	Meaning
20	Invalid character in fileid
24	Invalid filemode
28	File not found
36	Disk not accessed

FSWRITE

Use the FSWRITE macro instruction to write a record from an I/O buffer to a CMS disk file.

The format of the FSWRITE macro instruction is:

[label]	FSWRITE	{fileid [,FSCB=fscb]} [,ERROR=erraddr] {FSCB=fscb } [,FORM=E] [,options]

where:

label
is an optional statement label.

fileid
specifies the CMS file identifier. It may be:

'fn ft fm'
the fileid enclosed in single quotation marks and separated by blanks. If fm is omitted, A1 is assumed.

(reg)
a register other than 0 or 1 containing the address of the fileid (18 characters). When register format is used, the fileid must be exactly 18 characters in length; 8 for the filename, 8 for the filetype, and 2 for the filemode. Shorter names must be filled with blanks.

FSCB=*fscb*
specifies the address of an FSCB. It may be:

label
the label on an FSCB macro instruction.

(reg)
a register containing the address of an FSCB.

ERROR=*erraddr*
specifies the address of an error routine to be given control if an error is found. If ERROR= is not coded and an error occurs, control returns to the next sequential instruction in the calling program, as it does if no error occurs.

FORM=E
must be specified when the extended format FSCB is being used.

Options:

You can specify any of the following FSCB macro options on the FSWRITE macro instruction:

BUFFER=buffer
RECNO=number
BSIZE=size
NOREC=numrec
RECFM=format

These options may be specified either as the actual value (for example, NOREC=1) or as a register that contains the value (for example, NOREC=(3) where register 3 contains the value 1).

When you use any of these options, the associated field in the FSCB for the file is filled in or modified.

Usage Notes:

1. If an FSCB macro instruction has not been coded for a file (and the FSCB= operand is not coded on the FSWRITE macro instruction), you must specify the BUFFER= and BSIZE= options to indicate the location of the read/write buffer and the length of the record to be written. For the filemode, you must specify both a letter (A-Z) and a number (0-6). If the file is a variable-length file, you must also specify RECFM=V.

2. On return from the FSWRITE macro, register 1 contains the address of the FSCB for the file. If no FSCB exists, one is created following the FSWRITE macro instruction.

3. If you specify both fileid and FSCB=, the fileid is used to fill in the FSCB.

4. If the RECNO option is specified (either on the FSWRITE macro instruction or in the FSCB), that specified record is written. Otherwise, the next sequential record is written. For new files, writing begins with record 1; for existing files, writing begins with the first record following the end of the file.

5. To write records sequentially, beginning with a particular record number, use the RECNO option to specify the first record to be written. On the next FSWRITE macro instruction, use RECNO=0 so that writing continues sequentially, following the first record written.

6. To write blocked records (valid for fixed-length files only), use the BSIZE and NOREC options to specify the block size and number of records per block, respectively. For example, to write 80-byte records into 800-byte blocks, you should specify BSIZE=800 and NOREC=10. The buffer you use must be at least 800 bytes long.

7. When you use the FSWRITE macro to update an existing file of variable-length records, the replacement record must be the same length as the original record. An attempt to write a record shorter or longer than the original record on a disk formatted with 512-, 1K-, 2K-, or 4K-byte block size results in truncation of the file at the specified record number with no error return codes. An attempt to write a record shorter or longer than the original record on a disk formatted as an 800-byte block size results in no change to the file and an error code of 27.

8. The "update-in-place" facility allows you to write blocks back to their previous location on disk. The "update-in-place" attribute of a CMS file is indicated by the filemode number 6. This only applies to files located on a 512-, 1K-, 2K-, or 4K-byte block formatted minidisk.

   Note: For a variable format file, "update-in-place" applies only if a record is replaced by a record with the same length.

9. The CMS file system does not support zero length records. The FSWRITE macro cannot be used to write records with a length of zero.

Error Conditions:

If an error occurs, register 15 contains one of the following error codes:

Code	Meaning
2	Invalid buffer address.
3	Permanent I/O Error.
4	First character of filemode is invalid or disk not accessed.
5	Second character of filemode is invalid.
6	Item number too large (more than 65,535) will not fit in a halfword, extended PLIST not specified.
7	Attempt to skip over unwritten variable-length item.
8	Buffer size missing or invalid.
9	File open for input (for an 800-byte formatted disk).
10	Maximum number of files per minidisk reached (3400 for an 800-byte formatted disk).
11	Record format not F or V.
12	Attempt to write on read-only disk.
13	Disk is full.
14	Number of bytes to be written is not integrally divisible by the number of records to be written.
15	Length of fixed-length item not the same as previous item.
16	Record format specified not the same as file.
17	Variable-length item greater than 65535 bytes.
18	Number of records greater than 1 for variable-length file.
19	Maximum number of data blocks per file reached (16060 for an 800-byte formatted disk).
20	Invalid character detected in filename.
21	Invalid character detected in filetype.
22	Virtual storage capacity exceeded.
25	Insufficient free storage available for file directory buffers.

26	Requested item number is negative or item number plus number of items exceeds file system capacity.
27	Attempt to update variable length item with one of different length.

HNDEXT

Use the HNDEXT macro instruction to trap external interruptions and pass control to an internal routine for processing. In a virtual machine, external interruptions are caused by the CP EXTERNAL command. The format of the HNDEXT macro instruction is:

[label]	HNDEXT	{SET,address / CLR}

where:

label
is an optional statement label.

SET
specifies that you want to trap external interruptions.

address
specifies the address in your program of the routine to be given control when an external interruption occurs.

CLR
specifies that you no longer want to trap external interruptions.

Usage Notes:

1. External interruptions (other than timer interruptions) normally place your virtual machine in the debug environment.

2. When your interruption handling routine is given control, all virtual interruptions, except multiplexer, are disabled. If you are using the CMS blip function, all blips are stacked.

3. You are responsible for providing proper entry and exit linkage for your interruption handling routine. When your routine receives control, register 1 points to a save area in the format:

	Displacement	
Label	Dec	Hex
GRS	0	0
FRS	64	40
PSW	96	60
UAREA	104	68
END	176	B0

GRS, FRS, and PSW refer to the general purpose register, floating point register, and the program status word, respectively, at the time of the

interrupt. Register 13 points to the user save area at label UAREA. This save area is for your own use.

Register 15 contains the entry point address of your routine; it must return control to the address in register 14.

4. If you also issue a STAX macro instruction to handle attention interruptions while the HNDEXT macro is active, either exit may be interrupted while the other is running. If your exits depend on data in static areas, results are unpredictable.

5. If your program uses CMS IUCV support, IUCV external interrupts drive the exits set up by the HNDIUCV and CMSIUCV macros. In this case, the HNDEXT exit does not receive control. For more information on CMS IUCV support, see the *VM System Facilities for Programming*.

6. An STIMER exit will be taken before a HNDEXT exit if both exist in the same program and a timer interrupt occurs. The order of exits for a TIMER interrupt is STIMER, HNDEXT, and BLIP processing.

   Note: If a HNDEXT exit is coded and BLIP is SET ON, the HNDEXT exit will trap the BLIP timer interrupt.

7. It is your responsibility to issue a HNDEXT CLR in the application that issued a HNDEXT SET,address. Under certain environments, no cleanup is performed by DMSINT after the execution of a program.

HNDINT

Use the HNDINT macro instruction to trap interruptions for a specified I/O device. The format of the HNDINT macro instruction is:

```
[label]   HNDINT   {SET,(dev1 ,addr ,cuu ,ASAP ) [,(dev2...)...]}
                                 0        WAIT
                   {CLR,(dev1) [,(dev2)[...]]                   }
                    [,ERROR=erraddr]
```

where:

label
is an optional statement label.

SET
specifies that you want to trap interruptions for the specified device.

dev
specifies a four-character symbolic name for the device whose interruptions are to be trapped.

addr
specifies the address in your program of the routine to be given control when the interruption occurs. An address of 0 indicates that interruptions for the device are to be ignored.

cuu
specifies the virtual device address, in hexadecimal, of the device whose interruptions are to be trapped.

ASAP
specifies that the routine at addr is to be given control as soon as the interruption occurs.

WAIT
specifies that the routine at addr is to be given control after the WAITD macro is issued for the device.

CLR
specifies that you no longer want to trap interruptions for the specified device. HNDINT CLR should not be issued from within the interruption handling routine.

ERROR=*erraddr*
specifies the address of an error routine to be given control if an error is found. If ERROR= is not coded and an error occurs, control returns to the next sequential instruction in the calling program, as it does if no error occurs.

Usage Notes:

1. I/O operations initiated by some forms of the DIAGNOSE instruction do not produce I/O interruptions and are not trapped by HNDINT. If the I/O operation initiated by DIAGNOSE does produce I/O interrupts, HNDINT will trap the interrupts if the device has been specified for HNDINT. For specific information about I/O operations initiated by the DIAGNOSE instruction, see the *VM System Facilities for Programming*.

2. In a single HNDINT macro instruction, you can define interruption handling routines for more than one device. The argument list for each device must be enclosed in parentheses and separated from the next list by a comma.

3. If you specify WAIT, the routine at the specified address in your program receives control when a WAITD macro instruction that specifies the same symbolic device name is issued. If the WAITD macro instruction has already been issued for the device when the interruption occurs, the routine at the specified address receives control immediately.

4. You are responsible for establishing proper entry and exit linkage for your interruption handling routine. When your routine receives control, the significant registers contain:

```
Registers Contents
0-1       I/O OLD PSW
2-3       CHANNEL STATUS WORD (CSW)
  4       ADDRESS OF INTERRUPTING DEVICE
 14       RETURN ADDRESS
 15       ENTRY POINT ADDRESS
```

   Your routine must return control to the address in register 14, and indicate, via register 15, whether processing is complete. A zero (0) in register 15 means that you are through handling the interruption; any nonzero return code indicates that you expect another interruption.

   Note: Please note that register 13 does *not* point to a savearea for your use.

5. The interruption handling routine that you code should not perform any I/O operations. When it is given control, all I/O interruptions and external interruptions are disabled.

6. New applications should use the CONSOLE macro instead of the HNDINT macro to handle interrupts. Older applications may use HNDINT, but not in conjunction with CONSOLE.

   CONSOLE supports multiple applications for a 3270-type display device, while HNDINT supports only one. The CONSOLE macro should be used when doing I/O to a 3270-type device because the CONSOLE macro supersedes an HNDINT interrupt routine for the same device. An HNDINT interrupt routine will override a CONSOLE exit routine only in the case of an unsolicited interrupt. (See the CONSOLE macro.)

Error Conditions:

If an error condition occurs, register 15 will contain one of the following return codes:

Code	Meaning
1	Invalid device address (cuu) or interruption handling routine address (addr)
2	Trap item replaces another of same device name
3	Attempting to clear a nonexisting interruption

HNDSVC

Use the HNDSVC macro instruction to trap interruptions caused by specific supervisor call (SVC) instructions. The format of the HNDSVC macro instruction is:

[label]	HNDSVC	{SET,(svcnum,address) [,(svcnum,address)...]} {CLR,svcnum [,svcnum...]} [,ERROR=erraddr]

where:

label
is an optional statement label.

SET
specifies that you want to trap SVCs of the specified number(s).

svcnum
specifies the number of the SVC you want to trap. SVC numbers 0 through 200 and 206 through 255 are valid.

address
specifies the address of the routine in your program that should receive control whenever the specified SVC is issued.

CLR
specifies that you no longer want to trap the specified SVC(s).

ERROR=*erraddr*
specifies the address of an error routine to be given control if an error is found. If ERROR= is not coded and an error occurs, control returns to the next sequential instruction in the calling program, as it does if no error occurs.

Usage Note:

You are responsible for providing the proper entry and exit linkage for your SVC-handling routine. When your program receives control, the register contents are as follows:

```
Register  Contents
0-11      Same as when SVC instruction was issued
12        Address of your SVC-handling routine
13        Address of an 18-fullword save area (for your use)
14        Return address
15        Same as when SVC instruction was issued
```

Your routine need not restore any registers. Your routine must return control to the address in register 14.

Error Conditions:

If an error occurs, register 15 contains one of the following error codes:

Code	Meaning
1	Invalid SVC number, address, plist, or duplicate SVC numbers
2	SVC number set replaced previously set number
3	SVC number cleared was not set

IMMCMD

Use the IMMCMD macro instruction to declare, clear, and query Immediate commands. The four formats of the IMMCMD macro instruction are:

- Standard
- List (MF=L)
- Complex List (MF=(L,addr[,label]))
- Execute (MF=(E,addr))

Standard Format

The Standard Format of the IMMCMD macro instruction is:

[label]	IMMCMD	{SET NAME=command,EXIT=addr [,UWORD=addr] [,ERROR=addr] CLR NAME=command [,ERROR=addr] QRY NAME=command [,ERROR=addr]}

where:

label
is an optional statement label.

addr
is an assembler program label or an address stored in a general register. If a register is used, it must be enclosed in parentheses.

SET
establishes an Immediate command. If an Immediate command with the same name already exists, it is overridden in a stack-like manner.

CLR
clears an Immediate command. Any previously overridden Immediate command with the same name is reinstated by this action.

QRY
indicates that the caller is requesting information about an Immediate command. A return code from QRY indicates whether or not the Immediate command exists.

NAME=
is the name of the Immediate command. The command is specified as a 1- to 8-character word enclosed within single quotes or as an address stored in a register enclosed within parentheses. On SET, this is the name of the command being established. On CLR, this is the name of

the command being cancelled. On QRY, this is the name of the command that information is being sought for. This parameter is always required.

EXIT=

label
is an assembler program label that is the address of the exit routine.

(reg)
is a general register. Its value is the address of the exit routine.

EXIT is the routine that receives control when the command is entered from the terminal.

UWORD=

label
is an assembler program label that is the address that is stored as the UWORD.

(reg)
is a general register. Its contents are stored as the UWORD.

UWORD is an optional fullword that can be specified by the invoker for any purpose desired. When the exit routine gains control, UWORD is available to the exit.

ERROR=
specifies that the error routine receives control if an error is found. If you do not specify ERROR=, and an error occurs, control returns to the next sequential instruction (NSI) in the calling program, as it does if no error occurs.

label
is an assembler program label that is the address of the error routine.

(reg)
is a general register. Its value is the address of the error routine.

List Format

When MF=L is coded, the IMMCMD macro has the following format:

```
[label]   IMMCMD MF=L   [,NAME=command] [,EXIT=label] [,UWORD=label]
                        ,SET [,NAME=command] [,EXIT=label] [,UWORD=label]
                        ,CLR [,NAME=command]
                        ,QRY [,NAME=command]
```

The parameters have the same meaning as in the Standard Format except for the following:

MF=L
indicates that the parameter list is created in-line. No executable code is generated. Register notation cannot be used for macro parameter addresses.

Note: When you use the MF= parameter, all other parameters are optional. Before the function is executed, a valid combination of parameters must be specified by the List and Execute Formats of the macro.

Complex List Format

When MF=(L,addr[,label]) is coded, the IMMCMD macro has the following format:

```
[label]   IMMCMD MF=(L,addr[,label])   [,NAME=command][,EXIT=addr]
                                       [,UWORD=addr]
                                       ,SET[,NAME=command][,EXIT=addr]
                                           [,UWORD=addr]
                                       ,CLR[,NAME=command]
                                       ,QRY[,NAME=command]
```

The parameters have the same meaning as in the Standard Format except for the following:

MF=(L,addr[,label])
indicates that the parameter list is created in the area specified by addr. The address may be a label or may be specified in a register, and it represents an area within your program or an area of free storage obtained by a system service. You can determine the size of the parameter list by coding the label operand. The macro expansion equates label to the size of the parameter list. This format of the macro produces executable code to move the data into the parameter list specified by addr. It does *not* generate the instruction to invoke the function. If you use this version of the List Format, you must execute it prior to any related invocation of the Execute Format.

Note: When you use the MF= parameter, all other parameters are optional. Before the function is executed, a valid combination of parameters must be specified by the List and Execute Formats of the macro.

Execute Format

When MF=(E,addr) is coded, the IMMCMD macro has the following format:

```
[label]  IMMCMD MF=(E,addr)  [,NAME=command][,EXIT=addr][,UWORD=addr]
                             [,ERROR=addr]
                             ,SET[,NAME=command][,EXIT=addr]
                                 [,UWORD=addr][,ERROR=addr]
                             ,CLR[,NAME=command][,ERROR=addr]
                             ,QRY[,NAME=command][,ERROR=addr]
```

The parameters have the same meaning as in the Standard Format except for the following:

MF=(E,addr)
indicates that instructions are generated to execute the IMMCMD function.

addr
is a label or an address stored in a register that represents the location of the parameter list. Information in the parameter list may be changed by specifying the appropriate operands on the macro.

Note: When you use the MF= parameter, all other parameters are optional. Before the function is executed, a valid combination of parameters must be specified by the List and Execute Formats of the macro.

Usage Notes:

1. All Immediate commands established by the IMMCMD macro can be explicitly cancelled. If these Immediate commands are not explicitly cancelled by the IMMCMD macro, they are cancelled automatically, either by returning to the CMS command environment (except when in CMS subset mode) or CMS abend recovery.

2. Immediate commands that are established by the NUCXLOAD command or the NUCEXT function cannot be cleared by the IMMCMD macro. You must use the NUCXDROP command or the NUCEXT CANCEL function.

3. The IMMCMD macro provides the capability to give control to an exit routine whenever a specific Immediate command is invoked. These exit routines receive control as an extension of CMS I/O interrupt handling. Therefore, they receive control with a PSW key of 0 and disabled for interrupts. The exit routine must not perform any I/O operations or issue any SVC's that result in I/O operations. In addition, the exit routine must not enable itself for interrupts. DIAGNOSE instructions can be used within the exit, but the exit routine must not enable itself for interruptions that may be caused by the DIAGNOSE (for example,

DIAGNOSE X'58'). On entry, the exit routine is passed the following information:

R0 Address of Immediate command line in extended PLIST format.
R1 Address of Immediate command line in Standard PLIST format. The high order byte of register 1 is set to X'06' to indicate that this routine was invoked as a result of an Immediate command.
R2 Address of the IMMBLOK. The IMMBLOK contains the user word and other relevant information. The format of the IMMBLOK is as follows:

Bytes	Information
0-3	Address of next IMMBLOK
4-7	User word
8-15	Command name
16-19	Reserved
20-23	Entry point address

R12 Entry address
R13 Scratch area address (12 doublewords)
R14 Return address
R15 Entry address

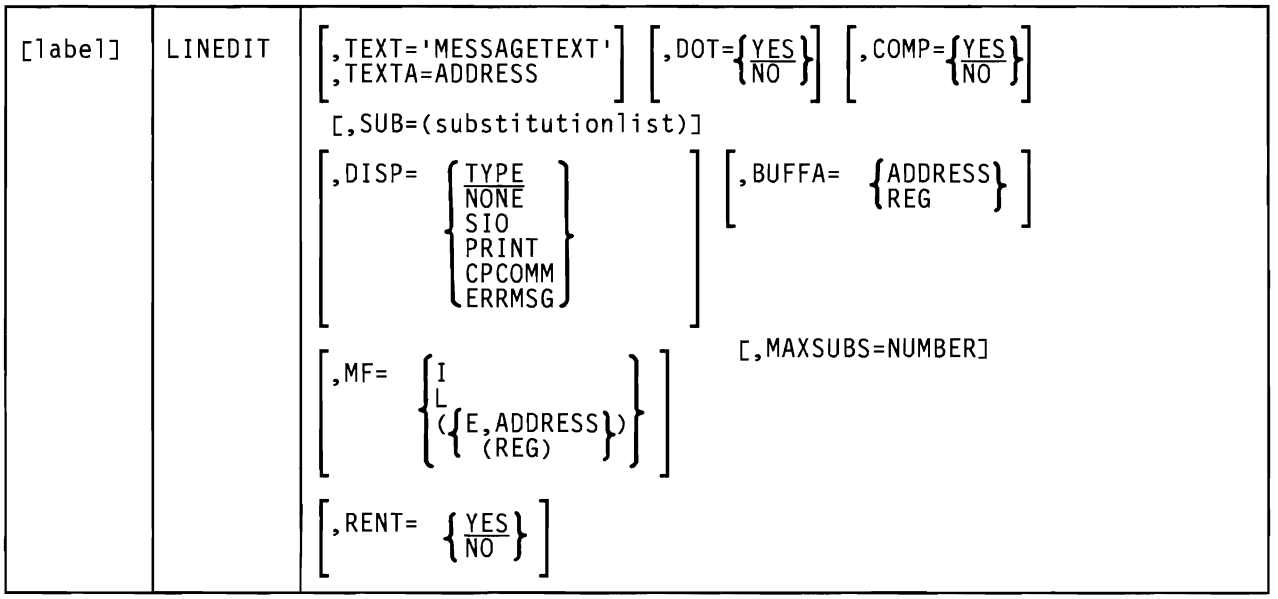
Error Conditions:

If an error occurs, register 15 contains one of the following return codes:

Code	Meaning
24	Invalid parameter list
44	Immediate command not found
48	Specified Immediate command is a nucleus extension and cannot be cleared
104	Not enough storage available to initialize the Immediate command

LINEDIT

Use the LINEDIT macro instruction to convert decimal values into EBCDIC or hexadecimal and to display the results at your terminal. The format of the LINEDIT macro instruction is:



The LINEDIT macro operands are listed below, briefly. For detailed formats, descriptions, and examples, refer to the appropriate heading following "LINEDIT Macro Operands."

TEXT='MESSAGE TEXT'
specifies the text of the message to be edited. The maximum length of the message text is 130 characters.

TEXTA=ADDRESS
specifies the address of the message text. It may be:

label
the symbolic address of the message text.

(*reg*)
a register containing the address of the message text.

DOT=
specifies whether a period is to be placed at the end of the line.

COMP=
specifies whether multiple blanks are to be removed from the line.

SUB=
specifies a substitution list describing the conversions to be performed on the line.

DISP=
specifies how the edited line is to be used. When DISP is not coded, the message text is displayed at the terminal.

BUFFA=
specifies the address of the buffer in which the line is to be copied.

MF=
specifies the macro format.

MAXSUBS=
specifies the maximum number of substitutions (MAXSUBS is used with the list form of the macro).

RENT=
specifies whether reentrant code must be generated.

Usage Notes:

1. You should never use registers 0, 1, or 15 as address registers when you code the LINEDIT macro instruction; these registers are used by the macro.

2. When message text for the LINEDIT macro instruction contains two or more consecutive periods, it indicates that a substitution is to be performed on that portion of the message. The number of periods you code indicates the number of characters that you want to appear as output. To indicate what values are to replace the periods, code a substitution list using the SUB operand.

3. When you use the standard (default) form of the LINEDIT macro instruction, reentrant code is produced, except when you specify more than one substitution list, or when you use register notation to indicate an address on the TEXTA or BUFFA operands. When any of these conditions occur, an MNOTE message is produced, indicating that the code is not reentrant.

   If you do not care whether the code is reentrant, you can specify the RENT=NO operand to suppress the MNOTE message. Otherwise, you can use the List and Execute Formats of the macro to write reentrant code (see "MF Operand").

4. When the macro completes, register 15 may contain a return code of 2 or 3, indicating that a channel 9 or channel 12 punch was sensed, respectively. You can use these codes to determine whether the end of the page is near (channel 9), or if the end of the page has been reached (channel 12). You might want to check for these codes if you want particular information at the bottom or at the end of each page being printed.

Error Conditions:

Errors can only occur if DISP=CPCOMM is specified. In this case, register 15 contains the return code from the CP command.

LINEDIT Macro Operands

TEXT Operand

Use the TEXT operand to specify the exact text of the message on the macro instruction. The message text must appear within single quotation marks, as follows:

```
TEXT='message text'
```

If you want a single quotation mark to appear within the actual message text, you must code two of them.

Text specified on the LINEDIT macro is edited so that multiple blanks appear as only a single blank, and a period is placed at the end of the line, for example:

```
LINEDIT TEXT='IT ISN''T     READY'
```

results in the display:

```
IT ISN'T READY.
```

TEXTA Operand

Use the TEXTA operand when you want to display a line that is contained in a buffer. You may specify either a symbolic address or use register notation, as follows:

```
TEXTA=  {label}
        {(reg)}
```

In either case, the first byte at the address specified must contain the length of the message text. For example:

```
LINEDIT TEXTA=MESSAGE
          .
          .
          .
MESSAGE  DC    X'16'
         DC    CL22'THIS IS A LINE OF TEXT'
```

If you use register notation with either the Standard or List Formats of the macro, the code generated is not reentrant. To suppress the MNOTE that informs you that code is not reentrant, use the RENT=NO operand.

DOT Operand

Use the DOT operand when you do not want a period placed at the end of the message text. The format of the DOT operand is:

```
DOT=  {YES}
      {NO }
```

For example, if you code:

```
LINEDIT TEXT='HI!',DOT=NO
```

the line is displayed as:

```
HI!
```

COMP Operand

Use the COMP operand when you want to display multiple blanks within your message text. The format of the COMP operand is:

```
COMP=  {YES}
       {NO }
```

For example, if you code:

```
 LINEDIT TEXT='TOTAL     5',COMP=NO
```

the line is displayed as:

```
TOTAL    5.
```

If COMP=YES, not only will all multiple blanks be reduced to single blanks, but any leading blanks will be removed as well.

SUB Operand

Use the SUB operand to specify the type of substitution to be performed on those portions of the message that contain periods. For each set of periods, you must specify the type of substitution and the value to be substituted or its address. The format of the SUB operand is:

```
SUB= ( { HEX {,(reg)     }                                  }      )
       { DEC {,expression}                                  }
       {                                                    }
       { HEXA{,address   }                                  }
       { HECA{,(reg)     }                                  }
       {                                                    }
       { HEX4A   {,address                    }             }
       { CHARA   {,(reg)                      }             }
       { CHARA8A {,({address},{length)}       }             }
       {          {(reg)   }  {(reg) }        }             }
```

Each of the possible substitution pairs is described below, followed by discussions of length specification and multiple substitution lists.

HEX,(*reg*)
converts the value in the specified register to graphic hexadecimal format and substitutes it in the message text. If you code fewer than eight consecutive periods in the message text, then leading digits are truncated; leading zeros are not suppressed.

For example, if register 3 contains the value C0031FC8, then the macro instruction:

```
LINEDIT TEXT='VALUE = ...',SUB=(HEX,(3))
```

results in the display:

```
VALUE = FC8.
```

HEX,*expression*
converts the given expression to graphic hexadecimal format and substitutes it in the message text. The expression may be a symbolic address or symbol equate; it is evaluated by means of a LOAD ADDRESS (LA) instruction. For example, if your program has a label BUFF1, the line:

```
LINEDIT TEXT='BUFFER IS LOCATED AT ......',SUB=(HEX,BUFF1)
```

might result in the display:

```
BUFFER IS LOCATED AT 0201AC.
```

If you code fewer than eight periods in the message text, leading digits are truncated; leading zeros are not suppressed.

DEC,(*reg*)
converts the value in the specified register into graphic decimal format and substitutes it in the message text. Leading zeros are suppressed. If the number is negative, a leading minus sign is inserted. For example, if register 3 contains the decimal value 10,345, then the macro instruction:

```
LINEDIT TEXT='REG 3 = ......',SUB=(DEC,(3))
```

results in the line:

```
REG 3 = 10345.
```

DEC,*expression*
converts the given expression to graphic decimal format and substitutes it in the message text. The expression may be a symbolic label in your program or a symbol equate. For example, if your program contains the statement:

```
VALUE    EQU    2003
```

then the macro instruction:

```
LINEDIT TEXT='VALUE IS ......',SUB=(DEC,VALUE+5)
```

results in the display:

```
VALUE IS 2008.
```

HEXA, *address*
converts the fullword at the specified address to graphic hexadecimal format and substitutes it in the message text. If you code fewer than eight periods in the message text, leading digits are truncated; leading zeros are not removed. For example, if you code:

```
LINEDIT TEXT='HEX VALUE IS .....',SUB=(HEXA,CODE)
```

then the last five hexadecimal digits of the fullword at the label CODE are substituted into the message text.

HEXA, (*reg*)
converts the fullword at the address indicated in the specified register into graphic hexadecimal format and substitutes it in the message text. For example, if you code:

```
LINEDIT TEXT='REGISTER 5 -> ......',SUB=(HEXA,(5))
```

then the last six hexadecimal digits of the fullword whose address is in register 5 are substituted in the message text.

If you code fewer than eight digits, leading digits are truncated; leading zeros are not suppressed.

DECA, *address*
converts the fullword at the specified address to graphic decimal format. Leading zeros are suppressed; if the number is negative, a minus sign is inserted. For example, if you code:

```
LINEDIT TEXT='COUNT = ......',SUB=(DECA,COUNT)
```

then the fullword at the location COUNT is converted to graphic decimal format and substituted in the message text.

DECA, (*reg*)
converts the fullword at the address specified in the indicated register into graphic decimal format and substitutes it in the message text. For example:

```
LINEDIT TEXT='SUM = ..........',SUB=(DECA,(3))
```

causes the value in the fullword whose address is in register 3 to be displayed in graphic decimal format.

HEX4A, *address*
converts the data at the specified address into graphic hexadecimal format, and inserts a blank character following every four bytes (eight characters of output). The data to be converted does not have to be on a fullword boundary. When you code periods in the message text for substitution, you must code sufficient periods to allow for the blanks.

For example, to display 8 bytes of information (16 hexadecimal digits), you must code 17 periods in the message text.

To display seven bytes of hexadecimal data beginning at the location STOR in your program, you could code:

```
LINEDIT TEXT='STOR: ..............',SUB=(HEX4A,STOR)
```

This might result in a display:

```
STOR: 0A23F115 78ACFE
```

Note that 15 periods were coded in the message text, to allow for the blank following the first four bytes displayed.

HEX4A, (*reg*)
converts the data at the address indicated in the specified register into graphic hexadecimal format and inserts a blank character following every four bytes displayed (eight characters of output).

When you code the message text for substitution, you must code sufficient periods to allow for the blank characters to be inserted.

For example, the line:

```
LINEDIT TEXT='BUFFER: ..................',SUB=(HEX4A,(6))
```

results in the display of the first nine bytes at the address in register 6, in the format:

```
hhhhhhhh hhhhhhhh hh
```

CHARA, *address*
substitutes the character data at the specified address into the message text. For example:

```
LINEDIT TEXT='NAME IS ''.........''',SUB=(CHARA,NAME)
```

causes the 10 characters at location NAME to be substituted into the message text. Multiple blanks are removed.

CHARA, (*reg*)
substitutes the character data at the address indicated in the specified register into the message text. For example:

```
LINEDIT TEXT='CODE IS ....',SUB=(CHARA,(7))
```

the first four characters at the address indicated in register 7 are substituted in the message line.

CHAR8A, *address*
substitutes the character data at the specified address into the message text, and inserts a blank character following each eight characters of output.

When you code the message text, you must code enough periods to allow for the blanks that will be substituted.

This substitution list is convenient for displaying CMS parameter lists. For example, to display a fileid in an FSCB, you might code

```
LINEDIT TEXT='FILEID IS ...................',
      SUB=(CHAR8A,OUTFILE+8)
```

where OUTFILE is the label on an FSCB macro. If the fileid for this file were TEST OUTPUT A1, then the LINEDIT macro instruction would result in the display:

```
FILEID IS TEST OUTPUT A1.
```

In the final edited line, multiple blanks are reduced to a single blank.

CHAR8A, (*reg*)
substitutes the character data at the address indicated in the specified register and inserts a blank character following each eight characters of output.

When you code the message text, you must include sufficient periods to allow for the blanks. For example:

```
LINEDIT TEXT='PLIST: ................................',
      SUB=(CHAR8A,(7))
```

results in a display of four doublewords of character data, beginning at the address indicated in register 7.

Specifying the Length for LINEDIT Macro Substitution: In all the examples shown, the length of the argument being substituted was determined by the number of periods in the message text. The number of periods indicated the size of the output field, and indirectly determined the size of the input data area.

For hexadecimal and decimal substitutions, the input data is truncated on the left. To ensure that a decimal number will never be truncated, you can code 10 periods (11 for negative numbers) in the message text where it will be substituted. For hexadecimal data, code eight periods to ensure that no characters are truncated when a fullword is substituted.

When you are coding substitution lists with the CHARA, CHAR8A, and HEX4A options, however, you can specify the length of the input data field. You must code the SUB operand as follows:

```
SUB=(type,(address,length))
```

Both address and length may be specified using register notation. For example:

```
SUB=(HEX4A,(LOC,(4)))
```

shows that the characters at location LOC are substituted into the message text; the number of characters is determined by the value contained in register 4, but it cannot be larger than the number of periods coded in the message text.

You can use this method in the special case where only one character is to be substituted. Since you must always code at least two periods to indicate that substitution is to be performed, you can code two periods and specify a length of one, as follows:

```
LINEDIT TEXT='INVALID MODE LETTER ..',SUB=(CHARA,(PLIST+24,1))
```

Specifying Multiple Substitution Lists: When you want to make several substitutions in the same line, you must enter a substitution list for each set of periods in the message text. For example:

```
LINEDIT TEXT='VALUES ARE ..... and ......',
           SUB=(DEC,(3),HEXA,LOC)
```

might generate a line as follows:

```
VALUES ARE -45 AND FFE3C2.
```

You should remember that if you are using the Standard Format of the macro instruction, and you want to perform more than one substitution in a single line, the LINEDIT macro will not generate reentrant code. If you code RENT=NO on the macro line, then you will not receive the MNOTE message indicating that the code is not reentrant. If you want reentrant code, you must use the List and Execute Formats of the macro instruction.

DISP Operand

Use the DISP operand to specify the output disposition of the edited line. The format of the DISP operand is:

```
DISP=  {TYPE  }
       {NONE  }
       {PRINT }
       {SIO   }
       {CPCOMM}
       {ERRMSG}
```

where:

DISP=TYPE
specifies that the message is to be displayed on the terminal. This is the default disposition.

DISP=NONE
specifies that no output occurs. This option is useful with the BUFFA operand.

DISP=SIO
specifies that the message is to be displayed, at the terminal, using SIO instead of TYPLIN, which is normally used. This option is used by CMS routines in cases where free storage pointers may be destroyed. Since lines are not stacked in the console buffer, no CONWAIT function is performed.

DISP=PRINT
specifies that the line is to be printed on the virtual printer. The first character of the line is interpreted as a carriage control character and does not appear on the printed output. (See the discussion of the PRINTL macro for a list of valid ASA control characters.) The maximum line size is 130 characters, including the ASA character.

When the macro completes, register 15 will contain a 2 if a channel 12 punch was sensed, or a 3 if a channel 9 punch was sensed. The location on the page being printed and the corresponding channel punch is defined by the current forms control buffer image being used. For information on how to specify the forms control buffer image for a virtual spooled printer, refer to the LOADVFCB and SPOOL commands in the *VM/SP CP Command Reference*. If you are using a virtual spooled 3800, refer to the CMS command SETPRT.

When the channel 9 or channel 12 punch is sensed, the write operation terminates after carriage spacing, but before writing the line. If you want to write the line without additional space, you must modify the carriage control character in the buffer to a code that writes without spacing (ASA code + or machine code 01).

You must issue the CP CLOSE command or the CP SPOOL PRT CLOSE command to close the virtual printer file. Issue the command either from your program (using an SVC 202 instruction or a LINEDIT macro instruction) or from the CMS environment after your program completes execution. The printer is automatically closed when you log off or when you use the CMS PRINT command.

DISP=CPCOMM
specifies that the line is to be passed to CP and executed as a CP command. For example:

```
LINEDIT TEXT='QUERY USERS',DISP=CPCOMM,DOT=NO
```

results in the CP command line being passed to CP and executed. On return, register 15 contains the return code from the CP command that was executed.

Note: When using the DISP=CPCOMM operand, specify DOT=NO (the default is YES).

DISP=ERRMSG
specifies that the line is to be checked to see if it qualifies for error message editing. If it does, it is displayed as an error message rather than as a regular line.

The standard header format of VM/SP error messages is:

```
xxxmmmnnns
```

where:

- xxxmmm is the name of the module issuing the message
- nnn is the message number
- s is the severity code

You can code whatever you want for the first nine characters of the code when you write error messages for your programs, but the tenth character must specify one of the following VM/SP message types:

Code	Message Type
I	Information
W	Warning
E	Error

The line is displayed according to the CP EMSG setting. If EMSG is set to:

- ON - the entire message is displayed
- TEXT - only the message portion is displayed
- CODE - only the 10-character code is displayed.

BUFFA Operand

Use the BUFFA operand to specify the address of a buffer into which the edited message is to be written. The message is copied into the indicated buffer, as well as being used as specified in the DISP operand. The format of the BUFFA operand is:

```
BUFFA=  {addr }
        {(reg)}
```

When the text is copied into the buffer, the length of the message text is inserted into the first byte of the buffer, and the remainder of the text is inserted in subsequent bytes.

If you use register notation to indicate the buffer address, the code generated will not be reentrant. To suppress the MNOTE that informs you that code is not reentrant, use the RENT=NO operand.

MF Operand

Use the MF operand to specify the macro format when you want to code List and Execute Formats when you write reentrant programs. The format of the MF operand is:

```
MF=  {I                  }
     {L                  }
     {(E,{addr }[,COPY])}
     {   {(reg)}       }
```

where:

MF=I (Standard Format)
generates an inline operand list for the LINEDIT macro instruction, and calls the routine that displays the message. This is the default. It generates reentrant code, except under the following circumstances:

- When you specify more than one substitution list
- When you use register notation with the TEXTA or BUFFA operands

MF=L (List Format)
generates a parameter list to be filled in when the Execute Format of the macro is used.

The size of the area reserved depends upon the number of substitutions to be made, which you can specify with the MAXSUBS operand. For example:

```
LINEDIT MF=L,MAXSUBS=5
```

reserves space for a parameter list that may hold up to five substitution lists. This same list may be used by several LINEDIT macro instructions.

MF=(E,addr) (Execute Format)
generates code to fill in the parameter list at the specified address, and calls the routine that displays the message text.

The address specified (either a symbolic address or in register notation) indicates the location of the List Format of the macro. The following example shows how you might use the List and Execute Formats of the LINEDIT macro to write reentrant code:

```
WRITETOT LINEDIT TEXT='SUBTOTAL ..... TOTAL .....',
                 SUB=(DEC,(4),DEC,(5)),MF=(E,LINELIST)
          .
          .
          .
LINELIST LINEDIT MF=L,MAXSUBS=6
```

When the Execute Format of the LINEDIT macro instruction is used, the parameter list for the message is built at label LINELIST, where the List Format of the macro was coded.

MAXSUBS Operand

Use the MAXSUBS operand when you code the List Format (MF=L) form of the LINEDIT macro instruction. The format of the MAXSUBS operand is:

```
MAXSUBS=number
```

where number specifies the maximum number of substitutions that will be made when the Execute Format of the macro is used.

RENT Operand

Use the RENT operand when you are going to use the Standard Format of the LINEDIT macro instruction and you do not care whether the code that is generated is reentrant. The format of the RENT operand is:

```
RENT=  {YES}
       {NO }
```

When RENT=YES (the default) is in effect, the LINEDIT macro expansion issues an MNOTE message indicating that nonreentrant code is being generated. This occurs when you use the Standard Format of the macro instruction and you specify one of the following:

- TEXTA=(reg)
- BUFFA=(reg)
- More than one substitution pair

Note: If you do not care whether the code is reentrant, and you do not wish to have the MNOTE appear, code RENT=NO. The RENT=NO coding merely suppresses the MNOTE statement; it has no effect on the expansion of the LINEDIT macro instruction.

LINERD

Use the LINERD macro instruction to read a line of input from the terminal. The LINERD macro can be used when CMS is running in full-screen mode (SET FULLSCREEN ON) or in linemode (SET FULLSCREEN OFF or SET FULLSCREEN SUSPEND). The four formats of the LINERD macro instruction are:

- Standard
- List (MF=L)
- Complex List (MF=(L,addr[,label]))
- Execute (MF=(E,addr))

Standard Format

The Standard Format of the LINERD macro is:

```
[label]   LINERD   DATA=(addr[,length])
                   [,VNAME=virtual screen name
                   ,LINE=addr,COL=addr
                   ,PROMPT=(addr[,length])|'text'
                   ,PAD=BLANK|NULL|NONE
                   ,LOGICAL=YES|NO
                   ,TRANS=YES|NO
                   ,CASE=UPPER|MIXED
                   ,TYPE=DIRECT|STACK|NOSTACK|INVISIBLE
                   ,WAIT=YES|NO
                   ,ATTREST=YES|NO
                   ,ERROR=addr]
```

where:

label
is an optional label for the statement.

DATA
specifies the address of a buffer into which the data is to be read and the length of this buffer. The address may be specified as an assembler program label or general register (2-12), enclosed in parentheses. This is a required parameter. When register notation is used, the length of the data buffer must be specified. If a label is used, the length can also be specified. If it is not specified, the length associated with the label will be used. The length may be specified as an absolute expression or in a general register (2-12), enclosed in parentheses. If an absolute expression, the value of the expression is

the length of the data to be written in the buffer. If a register, specified in parentheses, the register holds the length of the data.

PROMPT
specifies that prompt information is to be written before the read is performed. The prompt data may be specified as either a string of characters enclosed in single quotation marks or as the address and length of an area of storage. The address may be specified as an assembler program label or general register (2-12), enclosed in parentheses. If the length is specified and an address is not, the prompt information is assumed to reside in the read buffer. When register notation is used, the length of the data must be specified. If a label is used, the length can also be specified. If it is not specified, the length associated with the label will be used. The length may be specified as an absolute expression or in a general register (2-12), enclosed in parentheses. If an absolute expression, the value of the expression is the length of the data to be written. If a register, specified in parentheses, the register holds the length of the data.

VNAME
Specifies the virtual screen to be read. It can be specified as a literal string, up to 8 characters, enclosed in quotes, or it may be specified as an assembler label or a general register (2-12), enclosed in parentheses, which contains the address of an 8-byte name. If the VNAME parameter is not specified, the CMS virtual screen is assumed.

LINE
Specifies the address of a fullword in storage where the virtual screen line of the data read is stored. The address can be expressed as an assembler program label or as a register (2-12), enclosed in parentheses. This information is not available if FULLSCREEN is OFF.

COL
Specifies the address of a fullword in storage where the virtual screen column of the data read is stored. The address can be expressed as an assembler program label or as a register (2-12) enclosed in parentheses. This information is not available if FULLSCREEN is OFF.

PAD
specifies if padding should be done. If padding is requested, the input data is padded to the length of the input buffer with either nulls or blanks. The default is BLANKS. If NONE is specified, no padding is requested and the contents of the receiving field will equal the data read padded with the previous contents of that field.

LOGICAL
If YES is specified, a newline character in the input data is interpreted as a logical end-of-line. Only the logical line is returned. If NO is specified, the newline characters are ignored and the entire line is returned. The default is YES.

TRANS
If YES is specified, the input data is translated according to the user input translate table, if any, defined by the SET INPUT command. The default is YES.

CASE
If UPPER is specified, the input data is translated to upper case. If MIXED is specified, the input data is left as inputted. The default is UPPER.

TYPE
Specifies how the read request can be satisfied, as follows (the default is STACK):

DIRECT specifies that the input line is to be read directly from the virtual machine console. The input queue associated with the virtual screen and the stack are bypassed.

STACK specifies that the request can be satisfied by a line from the program stack, if one is available, or from the input queue of the specified virtual screen if it is not empty, or directly from the console. Lines read from the program stack are not subject to user input translation or logical line editing.

NOSTACK specifies that the program stack is bypassed. The request will be satisfied either by a line from the virtual screen input queue or by issuing a read.

INVISIBLE just like DIRECT but the characters entered in response are not displayed on the console.

WAIT
If you want to differentiate between system reads and program reads during program execution, the WAIT parameter may be used to dictate the status area message. If WAIT is specified as NO, or allowed to default, there will be no distinction between program or system reads. The status area message will be: "Enter a command or press a PF/PA key." If WAIT is specified as YES, the status area message will show that your program has requested input (a program read). The status area message will be: "Enter your response in vscreen VNAME." The WAIT=YES option is ignored if either DIRECT or INVISIBLE is specified on the TYPE parameter. The default is NO.

ATTREST
Specifies whether an attention interruption during a read should result in a restart of a read operation. The default is YES.

ERROR
Specifies the address of an instruction where execution should resume if an error occurs in the processing of the LINERD request, expressed either as a label or general register (2-12), enclosed in parentheses. If this parameter is omitted, execution will resume at the next sequential instruction.

List Format

The List Format (MF=L) of the LINERD macro is:

```
[label]   LINERD MF=L   [ ,DATA=(addr[,length])
                          ,VNAME=virtual screen name
                          ,LINE=addr,COL=addr
                          ,PROMPT=(addr[,length])|'text'
                          ,PAD=BLANK|NULL|NONE
                          ,LOGICAL=YES|NO
                          ,TRANS=YES|NO
                          ,CASE=UPPER|MIXED
                          ,TYPE=DIRECT|STACK|NOSTACK|INVISIBLE
                          ,WAIT=YES|NO
                          ,ATTREST=YES|NO ]
```

where:

The parameters have the same meaning as in the Standard Format except for the following:

MF=L
indicates that the parameter list is created in-line. No executable code is generated. Register notation cannot be used for macro parameter addresses.

Note: When you use the MF= parameter, all other parameters are optional. Before the function is executed, a valid combination of parameters must be specified by the List and Execute Formats of the macro.

Complex List Format

The Complex List Format (MF=(L,addr[,label])) of the LINERD macro is:

```
[label]   LINERD MF=(L,addr[,label])   [,DATA=(addr[,length])
                                       ,VNAME=virtual screen name
                                       ,LINE=addr,COL=addr
                                       ,PROMPT=(addr[,length])|'text'
                                       ,PAD=BLANK|NULL|NONE
                                       ,LOGICAL=YES|NO
                                       ,TRANS=YES|NO
                                       ,CASE=UPPER|MIXED
                                       ,TYPE=DIRECT|STACK|NOSTACK|INVISIBLE
                                       ,WAIT=YES|NO
                                       ,ATTREST=YES|NO
                                       ]
```

where:

The parameters have the same meaning as in the Standard Format except for the following:

MF=(L,addr[,label])
indicates that the parameter list is created in the area specified by addr. The address may be specified as an assembler program label or general register (2-12), enclosed in parentheses. It represents an area within a program or an area of free storage obtained by a system service. The size of the parameter list can be determined by coding the label operand. The macro expansion equates label to the size of the parameter list. This format of the macro produces executable code to move the data into the parameter list specified by addr. It does not generate the instruction to invoke the function. If you use this version of the List Format, you must execute it prior to any related invocation of the Execute Format.

Note: When you use the MF= parameter, all other parameters are optional. No default parameters are assumed. Before the function is executed, a valid combination of parameters must be specified by the List and Execute Formats of the macro.

Execute Format

The Execute Format (MF=(E,addr)) of the LINERD macro is:

```
[label]   LINERD MF=(E,addr)   [,DATA=(addr[,length])
                               ,VNAME=virtual screen name
                               ,LINE=addr,COL=addr
                               ,PROMPT=(addr[,length])|'text'
                               ,PAD=BLANK|NULL|NONE
                               ,LOGICAL=YES|NO
                               ,TRANS=YES|NO
                               ,CASE=UPPER|MIXED
                               ,TYPE=DIRECT|STACK|NOSTACK|INVISIBLE
                               ,WAIT=YES|NO
                               ,ATTREST=YES|NO
                               ,ERROR=addr]
```

where:

The parameters have the same meaning as in the Standard Format except for the following:

MF=(E,addr)
indicates that instructions are generated to execute the LINERD function. The address specifies the location of the parameter list. It can be specified as an assembler program label or general register (2-12), enclosed in parentheses. Information in the parameter list may be changed by specifying the appropriate operands on the macro.

Note: When you use the MF= parameter, all other parameters are optional. No default parameters are assumed. Before the function is executed, a valid combination of parameters must be specified by the List and Execute Formats of the macro.

Usage Notes:

1. When the function completes, register 0 contains the number of characters read.

2. When the virtual screen name is CMS (which is also the default virtual screen name), the action taken by LINERD depends on the setting of full-screen CMS. If SET FULLSCREEN is ON, LINERD waits for input into the CMS virtual screen. When the LINERD function is executed with CMS in linemode (SET FULLSCREEN OFF or SET FULLSCREEN SUSPEND), LINERD calls the RDTERM function to do the read. To support the length parameter, RDTERM must be called with the EDIT=PHYS option. In this case, data padding is restricted to BLANK or NULL because the RDTERM function clears the entire data buffer prior to doing the read.

3. If SET FULLSCREEN is ON, the ATTREST operand is ignored for CMS. If CMS is in linemode (SET FULLSCREEN OFF OR SET FULLSCREEN SUSPEND), ATTREST=NO can only be used when

reading physical lines (LOGICAL=NO). When ATTREST=NO, an attention interruption during a read operation signals the end of the line and does not result in a restart of the read.

4. If the prompt parameters are used with LOGICAL=NO, the read buffer may not be used for the prompt data because the read buffer may be cleared prior to the execution of the function.

5. If LOGICAL=YES, the maximum length for a read is 240. If LOGICAL=NO, the maximum length is 2030 bytes.

Error Message:

171T Permanent console error; re-IPL CMS

Return Codes:

Upon completion of the requested function, register 15 contains one of the following return codes:

0 Function executed successfully.

4 Attention interrupt ended the read operation (can only happen if SET FULLSCREEN OFF for CMS and DMSCRD is called).

12 Function is not valid for the virtual screen specified.

24 The user did not specify the parameter list correctly.

28 Virtual screen does not exist.

89 Console is a 2741 typewriter terminal.

104 Insufficient storage was available to execute the requested function.

LINEWRT

Use the LINEWRT macro instruction to display a line of output at the terminal. The LINEWRT macro can be used when CMS is running in full-screen mode (SET FULLSCREEN ON) or in linemode (SET FULLSCREEN OFF or SET FULLSCREEN SUSPEND). The four formats of the LINEWRT macro instruction are:

- Standard
- List (MF=L)
- Complex List (MF=(L,addr[,label]))
- Execute (MF=(E,addr))

Standard Format

The Standard Format of the LINEWRT macro is:

```
[label]   LINEWRT   DATA=(addr[,length])
                    [,VNAME=name
                    ,LINE=line,COL=col
                    ,COLOR=BLUE|RED|PINK|GREEN|TURQUOISE|
                           YELLOW|WHITE|DEFAULT
                    ,HILITE=HIGH|NOHIGH
                    ,EXTHI=BLINK|REVVIDEO|UNDERLINE|NONE
                    ,PSS=A|B|C|D|E|F|0
                    ,PROTECT=YES|NO
                    ,PRIOR=YES|NO
                    ,NOCR=YES|NO
                    ,ALARM=YES|NO
                    ,ERROR=addr]
```

where:

label
is an optional label for the statement.

DATA
specifies the address and length of the text to be written. The address may be specified as an assembler program label or general register (2-12), enclosed in parentheses. This is a required parameter. When register notation is used, the length of the data must be specified. If a label is used, the length can also be specified. If it is not specified, the length associated with the label will be used. The length may be specified as an absolute expression or in a general register (2-12), enclosed in parentheses. If an absolute expression, the value of the

expression is the length of the data to be written. If a register, specified in parentheses, the register holds the length of the data.

VNAME
specifies a previously-defined virtual screen name to which the write is associated. It can be specified as a literal string, up to 8 characters, enclosed in quotes, or it may be specified as an assembler label or general register (2-12), enclosed in parentheses which contains the address of an 8-byte name. If this parameter is omitted, the output will be directed to the CMS message class, which will be displayed in the CMS virtual screen by default.

LINE
is an integer > = 0 representing where on the virtual screen the data will be written. Line may be specified as an absolute expression or in a general register (2-12), enclosed in parentheses. If in a register, the register contains the line number. If this parameter is omitted, the data will be written on the line following the last line previously written.

COL
is an integer > = 0 representing where on the virtual screen the data will be written. Col may be specified as an absolute expression or in a general register (2-12), enclosed in parentheses. If in a register, the register contains the column number. If this parameter is omitted, the data will be written in the first column of the virtual screen.

COLOR
specifies the color in which the data should be written. Only one color may be specified. If no color is specified, the default color for the virtual screen is used. (See DEFINE VSCREEN command) One of the following keywords may be specified: DEFAULT BLUE RED PINK GREEN TURQUOISE YELLOW or WHITE. The DEFAULT keyword signifies the default color of the physical device.

HILITE
specifies the highlighting attribute for the data to be written. HIGH indicates bright (or high intensity) and NOHIGH indicates normal intensity. If HILITE is not specified, the default highlighting for the virtual screen is used. (See DEFINE VSCREEN command)

EXTHI
specifies the extended highlighting attribute for the data to be written. Only one of the following keywords for extended highlighting may be specified:

BLINK, REVVIDEO, UNDERLINE or NONE.

If EXTHI is not specified, the default extended highlighting for the virtual screen is used. (See DEFINE VSCREEN command)

PSS
specifies which programmable symbol set is to be used for the data to be written. Valid symbol sets may be specified as 0, A, B, C, D, E or F. Only one PSS may be specified. If no PSS is specified, the default character set for the virtual screen is used. (See DEFINE VSCREEN command)

PROTECT
specifies whether or not the data to be written is protected (cannot be typed over) or non-protected (may be typed over). The keywords YES|NO can be used. If no protect attribute is specified, the default for the virtual screen is used. (See DEFINE VSCREEN command)

PRIOR
The keywords YES|NO can be used. YES indicates a priority write. If PRIOR=YES, this data will be written even if CMS halt typing is in effect. NO is the default.

NOCR
The keywords YES|NO can be used. NOCR=YES indicates no carriage return; the cursor is set in the column following the data written. If NOCR=NO, the cursor is set in the line following the data written. NO is the default.

ALARM
The keywords YES|NO can be used. If ALARM=YES, the alarm is sounded the next time I/O is performed. NO is the default.

ERROR
specifies the address of an instruction where execution should resume if an error occurs in the processing of the LINEWRT request, expressed either as a label or general register (2-12), enclosed in parentheses. If this parameter is omitted, execution will resume at the next sequential instruction.

List Format

The List Format (MF=L) of the LINEWRT macro is:

```
[label]   LINEWRT MF=L   [ ,DATA=(addr[,length])
                           ,VNAME=name
                           ,LINE=line,COL=col
                           ,COLOR=BLUE|RED|PINK|GREEN|TURQUOISE|
                                  YELLOW|WHITE|DEFAULT
                           ,HILITE=HIGH|NOHIGH
                           ,EXTHI=BLINK|REVVIDEO|UNDERLINE|NONE
                           ,PSS=A|B|C|D|E|F|0
                           ,PROTECT=YES|NO
                           ,PRIOR=YES|NO
                           ,NOCR=YES|NO
                           ,ALARM=YES|NO ]
```

where:

The parameters have the same meaning as in the Standard Format except for the following:

MF=L
indicates that the parameter list is created in-line. No executable code is generated. Register notation cannot be used for macro parameter addresses.

NOTE: When you use the MF= parameter, all other parameters are optional. Before the function is executed, a valid combination of parameters must be specified by the List and Execute Formats of the macro.

Complex List Format

The Complex List Format (MF=(L,addr[,label])) of the LINEWRT macro is:

```
[label]   LINEWRT MF=(L,addr[,label])   [,DATA=(addr[,length])
                                        ,VNAME=name
                                        ,LINE=line,COL=col
                                        ,COLOR=BLUE|RED|PINK|GREEN|TURQUOISE|
                                               YELLOW|WHITE|DEFAULT
                                        ,HILITE=HIGH|NOHIGH
                                        ,EXTHI=BLINK|REVVIDEO|UNDERLINE|NONE
                                        ,PSS=A|B|C|D|E|F|0
                                        ,PROTECT=YES|NO
                                        ,PRIOR=YES|NO
                                        ,NOCR=YES|NO
                                        ,ALARM=YES|NO
                                        ]
```

where:

The parameters have the same meaning as in the Standard Format except for the following:

MF=(L,addr[,label])
indicates that the parameter list is created in the area specified by addr. The address may be specified as an assembler program label or general register (2-12), enclosed in parentheses. It represents an area within a program or an area of free storage obtained by a system service. The size of the parameter list can be determined by coding the label operand. The macro expansion equates label to the size of the parameter list. This format of the macro produces executable code to move the data into the parameter list specified by addr. It does not generate the instruction to invoke the function. If you use this version of the List Format, you must execute it prior to any related invocation of the Execute Format.

NOTE: When you use the MF= parameter, all other parameters are optional. No default parameters are assumed. Before the function is executed, a valid combination of parameters must be specified by the List and Execute Formats of the macro.

Execute Format

The Execute Format (MF=(E,addr)) of the LINEWRT macro is:

```
[label]   LINEWRT MF=(E,addr)   [,DATA=(addr[,length])
                                ,VNAME=name
                                ,LINE=line,COL=col
                                ,COLOR=BLUE|RED|PINK|GREEN|TURQUOISE|
                                       YELLOW|WHITE|DEFAULT
                                ,HILITE=HIGH|NOHIGH
                                ,EXTHI=BLINK|REVVIDEO|UNDERLINE|NONE
                                ,PSS=A|B|C|D|E|F|0
                                ,PROTECT=YES|NO
                                ,PRIOR=YES|NO
                                ,NOCR=YES|NO
                                ,ALARM=YES|NO
                                ,ERROR=addr                    ]
```

where:

The parameters have the same meaning as in the Standard Format except for the following:

MF=(E,addr)
indicates that instructions are generated to execute the LINEWRT function. The address specifies the location of the parameter list. It can be specified as an assembler program label or general register (2-12), enclosed in parentheses. Information in the parameter list may be changed by specifying the appropriate operands on the macro.

NOTE: When you use the MF= parameter, all other parameters are optional. No default parameters are assumed. Before the function is executed a valid combination of parameters must be specified by the List and Execute Formats of the macro.

Usage Notes:

1. For Color, Hilite, Exthi, and PSS, the user must be using a device which supports these characteristics; otherwise, they will be ignored.

2. When the virtual screen name is CMS (which is also the default virtual screen name), the action taken by LINEWRT depends on the setting of full-screen CMS. If SET FULLSCREEN is ON, LINEWRT writes the data into the CMS virtual screen. If CMS is in linemode (SET FULLSCREEN OFF or SET FULLSCREEN SUSPEND), LINEWRT calls the WRTERM function to display the output. In this case (writing to the CMS virtual screen in linemode CMS), the LINE, COL, COLOR, HILITE, EXTHI, PSS, and PROTECT parameters will be ignored.

   When the virtual screen name is not CMS, the setting of full-screen CMS has no effect on the LINEWRT macro.

3. The maximum length of data to be written is the size of the virtual screen being written to.

4. If the data to be written contains a X'15' anywhere in the text, and you are writing in line zero, the X'15' is treated as a linend character. Consequently, any text following the X'15' will be written in the next line.

Return Codes:

Upon completion of the requested function, register 15 contains one of the following return codes:

0 Function executed successfully.

12 Function is not valid for the virtual screen specified.

24 The user did not specify the parameter list correctly.

28 Virtual screen is not defined.

32 The specified line or column is outside the virtual screen.

104 Insufficient storage was available to execute the requested function.

PARSECMD

Use the PARSECMD macro instruction from an assembler program to parse (and translate) the arguments of a command.

There are four formats of the PARSECMD macro instruction:

- Standard
- List (MF=L)
- Complex List (MF=(L,addr[,label]))
- Execute (MF=(E,addr))

Standard Format

The Standard Format of the PARSECMD macro instruction is:

```
[label]  PARSECMD  UNIQID=uniqueid
                  [,APPLID=['DMS'|applid]]
                  [,PLIST=[(1)|addr]]
                  [,EPLIST=[(0)|addr]]
                  [,MSGDISP=[ERRMSG|NONE|EXECCOMM|var]]
                  [,MSGBUFF=[0|addr]]
                  [,TRANSL=[CMS|YES|NO|SAME]]
                  [,TYPCALL=[SVC|BALR]]
                  [,ERROR=addr]
```

where:

label
is an optional label for the statement.

UNIQID=
is the unique identifier of the syntax definition to be used for parsing. It has a maximum length of 16 characters and is always required.

APPLID=
is an application identifier such as DMS or OFS. It must be three alphanumeric characters, and the first character must be alphabetic. The default is DMS, which is the application identifier for CMS.

PLIST=
specifies the address of the tokenized plist for the command. The value specified may be a label or a general register (2-12), enclosed in parentheses. If the value is not specified, it is assumed that register 1 contains the address of the tokenized plist.

The high order byte of the address should indicate whether an extended plist is available at execution time. See *VM/SP CMS for System Programming* for the allowed values of this byte.

EPLIST=
specifies the address of the extended plist for the command. The value specified may be a label or a general register (2-12), enclosed in parentheses. If the value is not specified, it is assumed that register 0 contains the address of the extended plist.

MSGDISP=
is the disposition for parsing facility error messages. The default is ERRMSG.

ERRMSG
specifies that parser error messages will be written to the terminal according to the current setting of CP SET EMSG.

NONE
specifies that no output occurs and is most useful when used with the MSGBUFF option.

EXECCOMM
specifies that the message is to be returned to a variable in the EXEC calling this module. The complete message is copied into the variable 'MESSAGE', with the first line in 'MESSAGE.1', the second in 'MESSAGE.2', and so on. The number of lines in the message is copied into 'MESSAGE.0'. This can only be used when the module issuing PARSECMD is called from an EXEC.

var
specifies a variable defining the message display format to be used. The variable must be one byte long, and the low order three bits of the byte must be set to the desired disposition as follows:

```
ERRMSG   = 000
NONE     = 010
EXECCOMM = 100
```

MSGBUFF=
is the buffer for error message text. The default is zero and indicates no buffer. When the text is copied into the buffer, the length of the message occupies the first byte of the buffer, preceding the text.

Note: The length of the buffer, not including the length byte, must be placed in the first byte of the buffer before the call to PARSECMD is made.

TRANSL=
specifies whether the parsing facility should translate any keywords found in the parameter list.

CMS
specifies CMS will determine translation status for you, based on how the module issuing PARSECMD was invoked. This is the default; use this unless your program performs its own command resolution.

CMS uses YES if the specified command name was a translation (or a synonym or abbreviation of a translation) of the command invoked. NO is used if the specified command name was a synonym (or an abbreviation of a synonym), set with the SYNONYM command, of the command invoked. When a command is invoked with the same name it was specified by, SAME is used. See the *VM/SP CMS Command Reference* for additional information on how and when CMS will translate or synonym a command name.

YES
specifies that all keywords should be translated by the parsing facility. In other words, only keywords defined as national language (nl)-names in the Definition Language for Command Syntax (DLCS) syntax definition will be recognized.

NO
specifies keywords should not be translated by the parsing facility. In other words, only keywords defined as system language (sl)-names in the DLCS syntax definition will be recognized.

SAME
specifies the parsing facility should determine translation status from the first keyword found whose nl-name and sl-name DLCS definitions are different. This status is then used for any remaining keywords.

TYPCALL=
specifies how the parsing facility is to be called. The default is SVC.

SVC
indicates the parsing facility should be called via SVC 202.

BALR
indicates the parsing facility should be called via BALR 14,15. Register 13 must point to an 18 fullword savearea.

ERROR=
specifies that the error routine receives control if an error is found. If you do not specify ERROR=, and an error occurs, control returns to the next sequential instruction (NSI) in the calling program, as it does if no error occurs.

Note: Some IBM-supplied commands also use the PARSFLG parameter for special purposes. *Do not* use this parameter yourself.

PARSECMD

List Format

The List Format (MF=L) of the PARSECMD macro instruction is:

[label]	PARSECMD MF=L	[,UNIQID=uniqueid] [,APPLID=['DMS'|applid]] [,PLIST=addr] [,EPLIST=addr] [,MSGDISP=[ERRMSG|NONE|EXECCOMM|var]] [,MSGBUFF=[0|addr]] [,TRANSL=[CMS|YES|NO|SAME]]

The parameters have the same meaning as in the Standard Format except for the following:

MF=L
indicates a control block for invoking the parsing facility is created in-line. This control block is mapped by the PARSERCB macro. No executable code is generated. Register notation cannot be used for macro parameter addresses.

Note: When you use the MF=L parameter, all other parameters are optional. All non-specified parameters, except UNIQID, PLIST, and EPLIST, will default. Before the function is executed, a valid combination of parameters must be specified by the List and Execute Formats of the macro.

Complex List Format

The Complex List Format (MF=(L,addr[,label])) of the PARSECMD macro instruction is:

```
[label]  PARSECMD MF=(L,addr[,label])  [,UNIQID=uniqueid]
                                       [,APPLID=applid]
                                       [,PLIST=addr]
                                       [,EPLIST=addr]
                                       [,MSGDISP=[ERRMSG|NONE|EXECCOMM|var]]
                                       [,MSGBUFF=[0|addr]]
                                       [,TRANSL=[CMS|YES|NO|SAME]]
```

The parameters have the same meaning as in the Standard Format except for the following:

MF=(L,addr[,label])
indicates that the control block (a PARSERCB) is initialized in the area specified by addr. The address may be specified as an assembler program label or general register (2-12), enclosed in parentheses. It represents an area within a program or an area of free storage obtained by a system service. The size of the control block can be determined by coding the label operand. The macro expansion equates label to the size of the control block. This format of the macro produces executable code to move the data into the control block specified by addr. It does not generate the instruction to invoke the function. If you use this version of the List Format, you must execute it prior to any related invocation of the Execute Format.

Note: When you use the MF= (L,addr[,label]) parameter, all other parameters are optional. No default parameters will be assumed. Before the function is executed, a valid combination of parameters must be specified by the List and Execute Formats of the macro.

Execute Format

The Execute Format (MF=(E,addr)) of the PARSECMD macro instruction is:

```
[label]  PARSECMD MF=(E,addr)  [,UNIQID=uniqueid]
                               [,APPLID=applid]
                               [,PLIST=addr]
                               [,EPLIST=addr]
                               [,MSGDISP=[ERRMSG|NONE|EXECCOMM|var]]
                               [,MSGBUFF=[0|addr]]
                               [,TRANSL=[CMS|YES|NO|SAME]]
                               [,TYPCALL=[SVC|BALR]
                               [,ERROR=addr]
```

The parameters have the same meaning as in the Standard Format except for the following:

MF=(E,addr)
indicates that instructions are generated to execute the PARSECMD function. The address specifies the location of the control block. It can be specified as an assembler program label or general register (2-12), enclosed in parentheses. Information in the control block may be changed by specifying the appropriate operands on the macro.

Note: When you use the MF=(E,addr) parameter, all other parameters are optional. No default parameters, except TYPCALL=SVC, will be assumed. Before the function is executed, a valid combination of parameters must be specified by the List and Execute Formats of the macro.

Usage Notes:

1. The uniqueid you specify in the PARSECMD macro is matched up with the uniqueid specified in the Definition Language for Command Syntax (DLCS) file. See the *VM/SP CMS User's Guide* for a detailed description of uniqueids.

2. If you have not issued the MF=L format of this macro and you are not using a message buffer, code MSGBUFF=0.

3. On exit from the PARSECMD function, general register 1 contains the address of PARSERCB control block. Refer to PARSERCB macro and PVCENTRY macro for details on how to obtain the parsed and translated arguments.

4. When you invoke the PARSECMD macro (Standard and Execute formats), the parsing facility automatically obtains storage for the parsed (and translated) tokenized and extended plists and the PVCENTRY table. This storage will be automatically released when the module invoking PARSECMD returns to CMS. Do not try to free it yourself.

Error Conditions:

If an error occurs, register 15 contains one of the following codes:

Code	Meaning
24	Syntax error found
26	Application not active
28	Syntax definition not found in the command table or user function not found
104	Insufficient free storage

PARSERCB

Use the PARSERCB macro instruction to generate a DSECT for the PARSECMD control block.

The format of the PARSERCB macro instruction is:

[label]	PARSERCB

where:

label
is an optional label for the statement. The first statement in the PARSERCB macro expansion is labeled PARSERCB.

Usage Notes:

1. The PARSERCB macro instruction expands as follows:

```
PARSERCB    DSECT
PARNAME     DS      CL8          Parser entry point 'DMSPAR'
PARTOKIN    DS      AL4          Input tokenized plist address
PARTOKPT    DS      AL4          Parsed (translated) tokenized
*                                plist address
PAREPLIN    DS      AL4          Input extended plist address
PAREPLPT    DS      AL4          Parsed (translated) extended
*                                plist address
PARPTYPE    DS      XL1   F*1    Plist Type-High order byte of R1
PARTRANS    DS      XL1   F*2    Translation flag
PARTRYES    EQU     X'80'        Translation = YES (national lang)
PARTRNO     EQU     X'40'        Translation = NO (system lang)
PARTRSAM    EQU     X'20'        Translation = SAME (system=national)
PARSFLG     EQU     X'10'        Parsflg specified
PARMSG      DS      XL1   F*3    Message disposition
PARMSGER    EQU     X'00'        Message disposition is ERRMSG
PARMSGNO    EQU     X'02'        Message disposition is NONE
PARMSGXC    EQU     X'04'        Message disposition is EXECOMM
            DS      XL1   F*4    Reserved
PARPVCAD    DS      AL4          PVC table address
PARPVCNM    DS      F            Number of entries in PVC table
PARMSGAD    DS      AL4          Message buffer address
PARUNQID    DS      CL16         Syntax definition unique id
PARAPLID    DS      CL3          Application identifier
            DS      XL5          Reserved
PARLENBY    EQU     *-PARSERCB    Length of PARSERCB in bytes
PARLENDW    EQU     (PARLENBY+7)/8  Length of PARSERCB in dwords
```

2. The PARPVCAD field contains the address of the Parser Validation Code Table. Each entry in this table contains the address, length and validation code for a token in the parsed (and translated) extended plist (PAREPLPT). PARPVCNM gives the number of entries in this table; the entries are contiguous. Refer to the PVCENTRY macro for the mapping of each entry.

3. A PARSERCB is created by the Standard and List Formats of the PARSECMD macro and should be filled in with the other formats of the macro.

PARSERUF

Use the PARSERUF macro instruction to generate a mapping to the Parser Interface for User Token Validation Functions.

The format of the PARSERUF macro instruction is:

[label]	PARSERUF

where:

label
is an optional label for the statement. The first statement in the PARSERUF macro expansion is labeled PARSERUF.

Usage Notes:

1. The PARSERUF macro instruction expands as follows:

```
PARSERUF    DSECT
PARUNAME    DS      CL8             Name of function
PARUTKAD    DS      A               Address of token
PARUTKLG    DS      F               Length of token
PARUPVC     DS      XL1             User Function Validation Code
            DS      CL7             ** RESERVED **
PARUFNCE    DS      CL8'FF'         ** RESERVED **
PARUSZBY    EQU     *-PARSERUF      Length in bytes of this block
PARUSZDW    EQU     (PARUSZBY+7)/8  Length in DWORDS of this block
```

PRINTL

Use the PRINTL macro instruction to write a line or multiple lines to a virtual printer. The format of the PRINTL instruction is:

```
[label]   PRINTL   line [,length] [ ,CC={YES|NO|c} ] [ ,TRC={YES|NO|n} ]
                   [,FORM=([BUFFER] [,count] [,ccwaddr])]
                          [LIST]
                   [,CMSDEV=devaddr] [,ERROR=erraddr]
```

where:

label
is an optional statement label.

When writing one line to a virtual printer with each PRINTL instruction and **FORM=** is not specified,

line
specifies the line to be printed. It may be:

'linetext'
text enclosed in quotation marks.

lineaddr
the symbolic address of the line.

(reg)
a register (2-12) containing the address of the line.

length
specifies the length of the line to be printed. (See Usage Note 1.) It may be:

(reg)
a register (2-12) containing the length.

n
a self-defining term indicating the length.

When writing multiple lines to a virtual printer with each PRINTL instruction and **FORM=BUFFER**,

line
specifies the address of the buffer containing the fixed length records. It may be:

lineaddr
the symbolic address of the BUFFER.

(reg)
a register (2-12) containing the address of the BUFFER.

length
specifies the length of the records in the BUFFER. It may be:

(reg)
a register (2-12) containing the length.

n
a self-defining term indicating the length.

When writing multiple lines to a virtual printer with each PRINTL instruction and **FORM=LIST**,

line
specifies the address of the list of variable length records to be printed. It may be:

lineaddr
the symbolic address of the LIST.

(reg)
a register (2-12) containing the address of the LIST.

When FORM=LIST, the length of each record is specified in the LIST and the length parameter is ignored.

CC=
specifies whether or not the records to be printed contain a carriage control character in the first byte. The carriage control character specifies how many lines should be skipped before the next line is printed. It may be

YES
carriage control characters are in each line to be printed. YES is the default. See Usage Note 2.

NO
Carriage control characters are not present in the lines to be printed. If CC=NO is specified, the system will use the ASA carriage control character (X'40') to space one line before printing.

c

specifies an ASA carriage control character to be used for all lines. The lines to be printed are assumed not to contain carriage control characters. Refer to Usage Note 2 for valid ASA carriage control characters.

TRC=

specifies whether or not the current print line includes a TRC (Table Reference Character) byte. The TRC byte indicates which 3800 translate table is selected to print a line.

NO

specifies that there is no TRC byte in the line to be printed. NO is the default.

YES

specifies that the line to be printed has a TRC byte in the line. It is the second byte when a carriage control byte is present; otherwise, the TRC byte is the first byte. The value of the TRC byte determines which 3800 translate table is selected. If an invalid value is found, translate table 0 is selected.

n

specifies a value for TRC to indicate which 3800 translate table should be selected before printing the line. The line to be printed does not contain a TRC byte. If an invalid value is specified, translate table 0 is selected.

The value of the TRC byte corresponds to the order in which you have loaded WCGMs (via the CHARS keyword on the SETPRT and SPOOL commands). Valid values for TRC are 0, 1, 2, and 3.

FORM=([BUFFER | LIST] [,*count*] [,*ccwaddr*])

specifies that multiple records are to be printed with each PRINTL instruction.

BUFFER

specifies that fixed length records are in a buffer. The address of the buffer is specified by the line parameter and the number of records in the buffer is specified by count. The length of the records is specified by the length parameter. If you specify **TRC=**, it applies to all records in the buffer. The linetext parameter cannot be used. BUFFER is the default.

LIST

specifies that the addresses of variable length records are in a list. The address of the list is specified by the line parameter and the number of entries in the list is specified by count. The length of each record is specified in the list and the length parameter is ignored. If you specify **TRC=**, it applies to all records in the list. The linetext parameter cannot be used.

Each entry in the list is on a fullword boundary and contains 8 bytes:

Bytes	Information
0-3	Record address
4-5	Reserved
6-7	Record length

count
specifies the number of records to be printed. When FORM=BUFFER, it specifies the number of records in the BUFFER. When FORM=LIST, it specifies the number of entries in the LIST. The maximum number of records that can be printed with a single PRINTL request is 32,767. It may be:

n
a self-defining term indicating the number. The default is 55.

(reg)
a register (2-12) containing the number.

countaddr
the address of a halfword containing the number.

ccwaddr
specifies the address of a 4K buffer that is to be used to contain the CCW chains required to perform the requested I/O. If this parameter is not specified, the system will allocate a 4K buffer for you. To achieve optimum performance, specify this parameter. It may be:

label
a label of a register containing the symbolic address of the buffer.

(reg)
a register (2-12) containing the address.

CMSDEV=*devaddr*
specifies the 12-byte storage area containing the device characteristics provided by the CMSDEV macro. If not supplied, or if the contents of the area is zero, DIAGNOSE code X'24' is performed to determine the device type. It may be:

(reg)
a register (2-12) containing the address of the 12-byte area provided by the CMSDEV macro.

label
specifies the symbolic address of the 12-byte storage area provided by the CMSDEV macro.

ERROR=*erraddr*
specifies the address of an error routine to be given control if an error is found. If ERROR= is not coded and an error occurs, control returns to the next sequential instruction in the calling program, as it does if no error occurs.

label
is an assembler program label that is the address of the error routine.

(reg)
is a general register (2-12). Its value is the address of the error routine.

Usage Notes:

1. The maximum number of data bytes allowed is:

Virtual Printer Type	Maximum Data Bytes
1403	132
3203	132
3211	150
3800	204
4248	168

To determine the line length, add the following to your bytes of data:

- one byte for the carriage control character if CC=YES is specified,
- one byte for the TRC byte if TRC=YES is specified.

If you do not specify the length, it defaults to 133 characters, unless 'linetext' is specified. In this case, the length is taken from the length of the line text.

Lines which are greater than the carriage size will not be printed and a return code of 1 will be issued. However, lines with a carriage control character of X'5A' may have lengths up to 32767 bytes. If you use quoted data with a X'5A' carriage control, the line length must not be greater than 256 bytes.

2. When CC=YES, the first character of the line is interpreted as a carriage control character, which may be either ASA (ANSI) or machine code. The valid ASA control characters are:

Character	Hex Code	Meaning
blank	40	Space 1 line before printing
0	F0	Space 2 lines before printing
-	60	Space 3 lines before printing
+	4E	Suppress space before printing
1	F1	Skip to channel 1
2	F2	Skip to channel 2
3	F3	Skip to channel 3
4	F4	Skip to channel 4
5	F5	Skip to channel 5
6	F6	Skip to channel 6
7	F7	Skip to channel 7
8	F8	Skip to channel 8
9	F9	Skip to channel 9
A	C1	Skip to channel 10
B	C2	Skip to channel 11
C	C3	Skip to channel 12

Hex codes X'C1' and X'C3' are used in both machine code and ASA code. CMS recognizes these codes as ASA control characters, not as machine control characters.

Hex code X'5A' is recognized as only a machine code character. This code is used with a Composed Page Data Stream record.

When CC=NO, or when the line does not begin with a valid carriage control, the line is printed with an ASA carriage control character to space one line before printing (ASA X'40').

3. If you specify TRC= and the virtual printer is not a 3800, the TRC byte is stripped off before the line is printed. If the TRC byte is invalid, PRINTL issues the following MNOTE:

```
MNOTE 8,'INVALID TRC SPECIFICATION'
```

Translate table 0 is selected if the TRC byte is invalid.

4. For the CMSDEV= parameter, use the CMSDEV macro instruction to obtain printer characteristics and status.

5. When the macro completes, register 15 will contain a 2 if channel 12 was sensed, or a 3 if channel 9 was sensed. If the FORM= parameter is specified, channels 9 and 12 are ignored. When channel 9 or channel 12 is sensed, the write operation terminates after carriage spacing but before writing the line. If you want to write the line without additional space, you must modify the carriage control character in the buffer to a code that writes without spacing (ASA code + or machine code 01).

   The location on the page being printed and the corresponding channel is defined by the current forms control buffer image being used. For information on how to specify the forms control buffer image for a virtual spooled printer, refer to the LOADVFCB and SPOOL commands in the *VM/SP CP Command Reference*. If you are using a virtual 3800, also refer to the CMS SETPRT command.

6. You must issue the CP CLOSE command to close the virtual printer file. Issue the CLOSE command either from your program (using an SVC 202 instruction or a LINEDIT macro instruction) or from the CMS environment after your program completes execution. The printer is automatically closed when you log off or when you use the CMS PRINT command.

7. If the virtual printer is a 4248 with an extended FCB and the duplication option specified, you should check to be sure that the duplication offset contained in the extended FCB declaration is valid for the line length and that the line length is short enough to be duplicated. For more information on the extended FCB macro instruction, see *VM/SP CP for System Programming*.

Error Conditions:

If an error occurs, register 15 contains one of the following error codes:

Code	Meaning
1	Line too long
2	Channel 12 punch detected
3	Channel 9 punch detected
4	Intervention required
5	Unknown error
100	Printer not attached
104	Not enough storage available to successfully complete the program.

PUNCHC

Use the PUNCHC macro instruction to write a line to a virtual punch. The format of the PUNCHC macro instruction is:

[label]	PUNCHC	line [,ERROR=erraddr]

where:

label
is an optional statement label.

line
specifies the line to be punched. It may be:

'linetext'
text enclosed in single quotation marks.

lineaddr
the symbolic address of the line.

(reg)
a register containing the address of the line.

ERROR=*erraddr*
specifies the address of an error routine to be given control if an error is found. If ERROR= is not coded and an error occurs, control returns to the next sequential instruction in the calling program, as it does if no error occurs.

Usage Notes:

1. No stacker selecting is allowed. The line length must be 80 characters.
2. You must issue the CP CLOSE command to close the virtual punch file. Issue the CLOSE command either from your program (using an SVC 202 instruction) or from the CMS environment when your program completes execution. The punch is closed automatically when you log off or when you use the CMS PUNCH command.

Error Conditions:

If an error occurs, register 15 contains one of the following error codes:

Code	Meaning
2	Unit check
3	Unknown error
100	Punch not attached

PVCENTRY

Use the PVCENTRY macro instruction to generate a DSECT for the Parser Validation Code Table entry. Each Parser Validation Code Table entry contains the address, length and validation code for a token in the parsed (and translated) extended plist.

The format of the PVCENTRY macro instruction is:

[label]	PVCENTRY

where:

label
is an optional label for the statement. The first statement in the PVCENTRY macro expansion is labeled PVCENTRY.

Usage Notes:

1. The PVCENTRY macro instruction expands as follows:

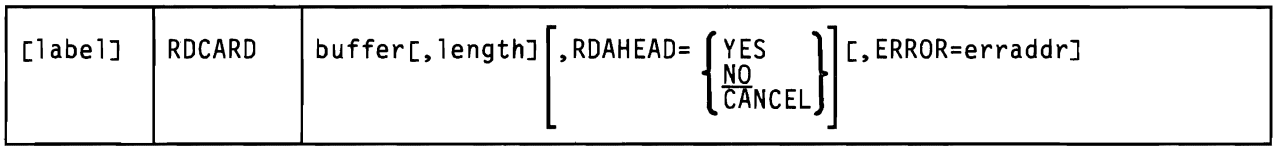
```
PVCENTRY  DSECT             Parser Validation Code Entry
PVCNEXTA  DS      A         Next PVC entry address, or 0 if last
PVCCODE   DS      XL1       Parser validation code
          DS      XL3       Reserved
PVCTTOKA  DS      A         Tokenized token address
PVCETOKA  DS      A         Extended token address
PVCETOKL  DS      F         Extended token length
*         EQU     X'00'     Reserved for IBM use
PVCCNAME  EQU     X'01'     Command Name
PVCKWORD  EQU     X'02'     Keyword
PVCOPTST  EQU     X'03'     Option start (
PVCOPTEN  EQU     X'04'     Option end )
PVCCOMMT  EQU     X'05'     Comment
PVCALNUM  EQU     X'06'     Alphanumeric string
PVCCHAR   EQU     X'07'     A single character
PVCCUU    EQU     X'08'     Device address:X'001',X'002',...,X'FFF'
PVCFN     EQU     X'09'     Filename
PVCFT     EQU     X'0A'     Filetype
PVCEFN    EQU     X'0B'     Filename with '*'
PVCEFT    EQU     X'0C'     Filetype with '*'
PVCEXECN  EQU     X'0D'     Execname
PVCEXECT  EQU     X'0E'     Exectype
PVCFM     EQU     X'0F'     Filemode
PVCHEX    EQU     X'10'     Hexadecimal number
PVCINT    EQU     X'11'     Integer: ..., -2, -1, 0, 1, 2, ...
PVCNINT   EQU     X'12'     Negative integer: ..., -2, -1
PVCPINT   EQU     X'13'     Positive integer: 1, 2, ...
PVCMODE   EQU     X'14'     Alphabetic character
PVCSTRIN  EQU     X'15'     Any character string(no blanks)
PVCTEXT   EQU     X'16'     Any string
PVCDIGIT  EQU     X'17'     Any unsigned integer
PVCAPPID  EQU     X'18'     Application identifier
```

```
PVCARBMD  EQU    X'19'       Arbitrary modifier
*         EQU    X'1A'-X'7E' Reserved for IBM use
PVCINVLD  EQU    X'7F'       Unconditionally invalid
*         EQU    X'80'-X'FF' Reserved for customer use
```

2. The parsing facility creates a table containing contiguous PVCENTRY entries that is addressed by PARSERCB. See the PARSERCB macro for details.

RDCARD

Use the RDCARD macro instruction to read a line from a virtual reader. The format of the RDCARD macro instruction is:



where:

label
is an optional statement label.

buffer
specifies the buffer address into which the line is to be read. It may be:

bufaddr
the symbolic address of the buffer.

(reg)
a register (2-12) containing the address of the buffer.

length
specifies the length of card to be read. The minimum length is 80; the maximum length is 204. The default is 80. The length may be specified in one of two ways:

n
a self-defining term indicating the length.

(reg)
a register (2-12) containing the length.

RDAHEAD=
specifies whether or not as many lines as possible are to be read into an internal I/O buffer before each line is read into the specified buffer. It may be:

YES
reads multiple lines into an internal I/O buffer. See Usage Notes 5 and 6.

NO
will not read multiple lines into an internal I/O buffer. NO is the default.

CANCEL
releases the internal I/O buffer used for RDAHEAD=YES. Any lines in the buffer will be lost.

ERROR=*erraddr*
specifies the address of an error routine to be given control if an error is found. If ERROR= is not coded and an error occurs, control returns to the next sequential instruction in the calling program, as it does if no error occurs.

Usage Notes:

1. No stacker selecting is allowed.

2. When the macro completes, register 0 contains the length of the card that was read.

3. You may not use the RDCARD macro in jobs that run under the CMS batch machine.

4. If the reader file being processed contains carriage control characters, the RDCARD macro returns the records with the carriage control characters stripped off.

5. If you process RDCARD with RDAHEAD=YES and the virtual card reader is closed before an error condition is detected (other than wrong-length record, RC=5), lines may still remain in the buffer. Subsequent RDCARD calls return the next available lines from the internal buffer until it is empty. Changes in the status of the virtual card reader are not recognized until the buffer is empty and the next physical read is performed. For most applications that read to end-of-file, RDAHEAD=YES should be specified.

   To insure that the internal I/O buffer is released and that the next RDCARD request will read from the virtual reader, not the internal buffer, issue RDCARD with RDAHEAD=CANCEL and a length of zero.

6. RDAHEAD=NO is forced if the logical record length is greater than 2028, or if there is insufficient storage to allocate the internal I/O buffer.

Error Conditions:

If an error occurs, register 15 contains one of the following error codes:

Code	Meaning
1	End of file
2	Unit check
3	Unknown error
5	Length not equal to requested length
100	Device not attached

RDTAPE

Use the RDTAPE macro instruction to read a record from the specified tape drive. The format of the RDTAPE macro instruction is:

[label]	RDTAPE	buffer,length [,device] [,MODE=mode] [,ERROR=erradr]

where:

label
is an optional statement label.

buffer
specifies the buffer address into which the record is to be read. It may be specified in either of two ways:

lineaddr
the symbolic address of the buffer.

(reg)
a register (2-12) containing the address of the buffer.

length
specifies the length of the largest record to be read. A 65,535-byte record is the largest record that can be read. It may be specified in either of two ways:

n
a self-defining term indicating the length.

(reg)
a register (2-12) containing the length.

device
specifies the device from which the line is to be read. If omitted, TAP1 (virtual address 181) is assumed. It may be specified in either of two ways:

TAP*n*

cuu
indicates the symbolic tape identification (TAPn) or the virtual device address (cuu) of the tape to be read. The following symbolic names and virtual device addresses are supported.

Symbolic Name	Virtual Address	Symbolic Name	Virtual Address
TAP0	180	TAP8	288
TAP1	181	TAP9	289
TAP2	182	TAPA	28A
TAP3	183	TAPB	28B
TAP4	184	TAPC	28C
TAP5	185	TAPD	28D
TAP6	186	TAPE	28E
TAP7	187	TAPF	28F

MODE = *mode*
specifies the number of tracks, density, and tape recording technique options. It must be in the following form:

```
([track],[density],[trtch])
```

track

7 indicates a 7-track tape (implies density = 800 and trtch = O).

9 indicates a 9-track tape (implies density = 800).

18 indicates an 18-track tape (implies density = 38K).

density

200, 556, or 800 for a 7-track tape.

800, 1600, or 6250 for a 9-track tape.

38K for an 18-track tape.

trtch

indicates the tape recording technique for 7-track tape. One of the following must be specified:

O	odd parity, converter off, translator off.
OC	odd parity, converter on, translator off.
OT	odd parity, converter off, translator on.
E	even parity, converter off, translator off.
ET	even parity, converter off, translator on.

ERROR=*erraddr*
specifies the address of an error routine to be given control if an error is found. If ERROR= is not coded and an error occurs, control returns to the next sequential instruction in the calling program, as it does if no error occurs.

Usage Notes:

1. When the macro completes, register 0 contains the number of bytes read.

2. If the MODE option is not specified, the tape drive will use the default density of the drive. The default density for dual density drives is the highest density. This is true even if the TAPE MODESET command is entered right before the macro is issued.

3. You need not specify the MODE option when you are reading from a 9-track tape and using the default density of the tape drive nor when you are reading from a 7-track tape with a density of 800 bpi, odd parity, with the data converter and translator off.

4. You need not specify the mode option (track and/or density) when reading from a 3480 tape.

Error Conditions:

If an error occurs, register 15 contains one of the following error codes:

Code	Meaning
1	Invalid function or parameter list
2	End of file or end of tape
3	Permanent I/O error
4	Invalid device address
5	Tape not attached
7	Invalid device attached
8	Incorrect length error

RDTERM

Use the RDTERM macro instruction to read a line from the terminal into an I/O buffer. The format of the RDTERM macro instruction is:

[label]	RDTERM	buffer[,EDIT=code] [,LENGTH=length] [,PRBUFF=addr] [,TYPE=DIRECT] [,PRLGTH=length] [,ATTREST={YES|NO}]

where:

label
is an optional statement label.

buffer
specifies the address of a buffer into which the line is to be read. The buffer is assumed to be 130 bytes long, unless EDIT=PHYS is specified. The address may be specified as:

lineaddr
the symbolic address of the buffer.

(reg)
a register (2-12) containing the address of the buffer.

EDIT=*code*
specifies the type of editing, if any, to be performed on the input line.

LN
indicates that a logical line is to be read and no editing is to be done.

PAD
requests that the input line be padded with blanks to the length specified.

UPCASE
requests that the line be translated to uppercase.

YES
indicates both padding and translation to uppercase. YES is the default.

PHYS
indicates that a physical line is to be read. When PHYS is specified, the LENGTH and ATTREST = NO operands may also be entered. This option causes the input line to be translated using the user translation table.

LENGTH = *length*
specifies the length of the buffer. If not specified, 130 is assumed. The maximum length is 2030 bytes. The length may be specified only if EDIT = PHYS (see Usage Note 2). It may be specified in either of two forms:

n
a self-defining term indicating the length of the buffer

(reg)
a register (2-12) containing the length of the buffer.

PRBUFF = *addr*
Specifies the address of a buffer in which the prompt data resides. The length of the prompt data to be written is specified by the PRLGTH parameter. If the PRLGTH parameter is specified, but the PRBUFF parameter is not, the prompt information is assumed to reside in the read buffer. The PRBUFF address can be specified as follows:

addr
the symbolic address of the buffer.

(reg)
a register (2-12) containing the length of the buffer.

TYPE = DIRECT
indicates that the the input line is to be read directly from the virtual machine console. The terminal input buffer and the program stack are bypassed.

PRLGTH = *length*
Specifies the length of the prompt information to be written prior to the read. The prompt information is written with no carriage return. The prompt information is written from the user's read data buffer or from the buffer specified by the PRBUFF parameter. The length can be specified in either of two forms:

n
a self-defining term indicating the length of the buffer

(*reg***)**
a register (2-12) containing the length of the buffer.

ATTREST = YES NO
specifies whether an attention interruption during a read should result in a restart of the read operation. (See Usage Note 2.)

Usage Notes:

1. When the macro completes, register 0 contains the number of characters read.

2. You can use the ATTREST = NO and LENGTH operands only when you are reading physical lines (EDIT = PHYS). When ATTREST = NO, an attention interruption during a read operation signals the end of the line and does not result in a restart of the read. These operands are used primarily in writing VS APL programs.

   Note: If you are using a typewriter terminal, and specify ATTREST = NO, CMS restarts a read when an attention is generated on a null line. The only way to terminate the read is by pressing the carriage return.

3. The PRBUFF and PRLGTH operands are intended for use with TTY type devices. The maximum PRLGTH is 1760 characters. These operands are not supported for TTY devices attached via VCNA or PASSTHRU.

4. If the prompt parameters are used with EDIT = PHYS, the read buffer may not be used for the prompt data because the read buffer is cleared prior to the execution of the function.

5. Use the LINERD macro instead of the RDTERM macro for increased flexibility. The LINERD macro supports all of the functions that RDTERM does, and it also provides enhanced input data editing and allows the user to specify a virtual screen name. (See the LINERD macro.)

Error Conditions:

If an error occurs, register 15 contains one of the following error codes:

Code	Meaning
2	Invalid parameter
4	Read was terminated by an attention signal (possible only when ATTREST = NO)

REGEQU

Use the REGEQU macro instruction to generate a list of EQU (equate) statements to assign symbolic names for the general, floating-point, and extended control registers. The format of the REGEQU macro instruction is:

	REGEQU	

Usage Note:

The REGEQU macro instruction causes the following equate statements to be generated:

```
General Registers          Extended Control Registers

R0      EQU   0            C0      EQU   0
R1      EQU   1            C1      EQU   1
R2      EQU   2            C2      EQU   2
R3      EQU   3            C3      EQU   3
R4      EQU   4            C4      EQU   4
R5      EQU   5            C5      EQU   5
R6      EQU   6            C6      EQU   6
R7      EQU   7            C7      EQU   7
R8      EQU   8            C8      EQU   8
R9      EQU   9            C9      EQU   9
R10     EQU   10           C10     EQU   10
R11     EQU   11           C11     EQU   11
R12     EQU   12           C12     EQU   12
R13     EQU   13           C13     EQU   13
R14     EQU   14           C14     EQU   14
R15     EQU   15           C15     EQU   15

Floating-Point Registers

F0      EQU   0
F2      EQU   2
F4      EQU   4
F6      EQU   6
```

SENDREQ

Programs use this macro to make service requests. You must provide the address of an initialized CPRB in a general purpose register. Control returns to the requesting program with an appropriate CMS return code in register 15 and the server return code field CRBSRTNC of the CPRB (see the *VM/SP IBM Programmer's Guide to the Server-Requester Programming Interface for VM/SP*).

The format of the SENDREQ macro instruction is:

[label]	SENDREQ	[CPRBREG=n]

where:

label
is an optional statement label.

CPRBREG=*n*
n is the name of a register containing the address of a properly initialized service request CPRB. If this parameter is not specified, register 1 (n=1) is assumed.

Note: The SENDREQ macro always uses Register 0 and Register 1.

The following assembly message (MNOTE) may be produced during assembler processing of the SENDREQ macro:

```
DMSMAC011E  CPRBREG OPERAND OMITTED IN SENDREQ MACRO
```

For more information on the SENDREQ macro and how to use it with Enhanced Connectivity Facilities on VM/SP, see the *VM/SP IBM Programmer's Guide to the Server-Requester Programming Interface for VM/SP*.

TAPECTL

Use the TAPECTL macro instruction to position the specified tape according to the specified function code. The format of the TAPECTL macro instruction is:

[label]	TAPECTL	function [,device][,MODE=mode][,ERROR=erraddr] [,BLKBUFF=blkadr]

where:

label
is an optional statement label.

function
specifies the control function to be performed. It must be one of the following codes:

Code	Function
REW	Rewind the tape
RUN	Rewind and unload the tape
ERG	Erase a defective section of the tape
BSR	Backspace one record
BSF	Backspace one file
FSR	Forward-space one record
FSF	Forward-space one file
WTM	Write a tape mark
LOCBLK	Locate block (3480 only)
RDBLKID	Read block ID (3480 only)

device
specifies the tape on which the control operation is to be performed. If omitted, TAP1 (virtual address 181) is assumed. It may be:

TAP*n*

cuu
indicates the symbolic tape identification (TAPn) or the virtual device address (cuu) of the tape to be positioned. The following symbolic names and virtual device addresses are supported:

Symbolic Name	Virtual Address	Symbolic Name	Virtual Address
TAP0	180	TAP8	288
TAP1	181	TAP9	289
TAP2	182	TAPA	28A
TAP3	183	TAPB	28B
TAP4	184	TAPC	28C
TAP5	185	TAPD	28D
TAP6	186	TAPE	28E
TAP7	187	TAPF	28F

MODE = *mode*

specifies the number of tracks, density, and tape recording technique options. It must be in the following form:

```
([track],[density],[trtch])
```

track

7 indicates a 7-track tape (implies density = 800 and trtch = O).

9 indicates a 9-track tape (implies density = 800).

18 indicates a 18-track tape (implies density = 38K).

density

200, 556, or 800 for a 7-track tape.

800, 1600, or 6250 for a 9-track tape.

38K for an 18-track tape.

trtch

indicates the tape recording technique for 7-track tape. One of the following must be specified:

O	odd parity, converter off, translator off.
OC	odd parity, converter on, translator off.
OT	odd parity, converter off, translator on.
E	even parity, converter off, translator off.
ET	even parity, converter off, translator on.

ERROR = *erraddr*

specifies the address of an error routine to be given control if an error is found. If ERROR= is not coded and an error occurs, control returns to the next sequential instruction in the calling program, as it does if no error occurs.

BLKBUFF = *blkadr*

Use this option only when the LOCBLK or RDBLKID function is specified.

- For RDBLKID, specify BLKBUFF=blkadr, where blkadr is the address of a location that contains 0.
- For LOCBLK, specify BLKBUFF=blkadr, where blkadr is the address of a location that contains a 4-byte block ID.

See Usage Notes 3 and 4.

Usage Notes:

1. If the MODE option is not specified, the tape drive will use the default density of the drive. The default density for dual density drives is the highest density. This is true even if the TAPE MODESET command is entered right before the macro is issued.
2. You need not specify the MODE option when you are performing any of the following operations:
   a. Manipulating a 9-track tape and you are using the default density for the tape system.
   b. Writing a 7-track tape with a density of 800 bpi, odd parity, with data converter and translator off.
   c. Reading from a 3480 tape.
3. A block ID identifies a block of data or a file mark for the Locate Block and Read Block ID functions of the 3480 Magnetic Tape Subsystem.
4. The RDBLKID function of the 3480 tape subsystem is used to obtain:
   a. The block ID to be used in the LOCBLK function. This is the ID of the next data block that will be passed between VM/SP and the 3480 Tape Control Unit in a read or write mode.
   b. The block ID of the next data block to be written to or read from the tape in read or write mode.
   c. The block ID of the data block that was last sent to VM/SP in read-backward mode.

d. The block ID of the data block that was last read into the control unit buffer from the tape drive in read-backward mode.

The RDBLKID function causes an 8-byte field to be returned to VM/SP. The field contains two 4-byte block IDs. For the RDBLKID function, specify BLKBUFF=blkadr, where blkadr is the address of a location that contains 0.

The first block ID (bytes 0-3) is called the physical block ID. The physical block ID identifies the following:

a. The data block that is about to be passed between VM/SP and the 3480 Tape Control Unit in read or write mode. This is the block ID usually specified by the option BLKBUFF=blkadr for the LOCBLK function.

b. The last data block that was sent to VM/SP in read-backward mode.

The first data block stored on the 3480 tape cartridge will have a block ID of X'01000000'.

The second block ID (bytes 4-7), called the logical block ID, identifies the data block that is:

a. The next to be written to the tape drive from the control unit buffer in write mode.

b. The next to be read from the tape drive to the buffer in read mode.

c. The last one read into the buffer from the tape drive in read-backward mode.

The control unit buffer is in write mode, read mode, or read-backward mode if VM/SP has been executing write, read, or read-backward commands, respectively, and the control unit anticipates the same type of command will continue to be executed. For example, if VM/SP executes a write command, the control unit buffer is set to write mode until another command, such as a read command, sets it to another mode. If no mode set is in the buffer (that is, the control unit is not performing any anticipatory buffering), the two block IDs returned by the RDBLKID function will be the same.

If VM/SP does not continue with the anticipated read, write, or read-backward commands, the first command that invalidates the anticipated mode causes the following to happen:

a. Data in the buffer that is to be written to tape is written to tape.

b. Data in the buffer that has been read from the tape, but not passed to VM/SP, is purged.

c. Tape is positioned to complete the command that caused the change of mode.

5. The LOCBLK function of the 3480 tape subsystem causes a 4-byte field to be sent to the control unit to position the tape so that VM/SP may forward-read the specified data block from the tape or write the specified data block to the tape. The address of the ID of the data block is specified with the BLKBUFF=blkadr option.

   For this function, the BLKBUFF=blkadr option normally contains the address of a location that contains the physical block ID (bytes 0-3) returned by the RDBLKID function. This is the ID of the next data block that is to be passed between VM/SP and the 3480 Tape Control Unit in read or write mode. In read-backward mode, it is the ID of the last data block that was sent to VM/SP.

   To position the tape at high speed to a point just past the last block written on a tape, you may specify a block ID that does not exist. This action is taken to position the tape to write data to it. You are responsible for knowing if a block ID exists. If a specified block ID cannot be found by the control unit, it assumes the tape is to be positioned for a write operation. You may search the tape for a block ID in either direction.

Error Conditions:

If an error occurs, register 15 contains one of the following error codes:

Code	Meaning
1	Invalid function or parameter list.
2	End of file or end of tape
3	Permanent I/O error
4	Invalid device id
5	Tape is not attached
6	Tape is file-protected
7	Invalid device attached

TAPESL

The TAPESL macro processes IBM standard HDR1 and EOF1 labels without using DOS or OS OPEN and CLOSE macros. This macro is used with RDTAPE, WRTAPE, and TAPECTL. TAPESL processes only HDR1 and EOF1 labels. It does not process other labels, such as standard user labels or HDR2 labels. It does not perform any functions of opening a tape file other than label checking or writing. The same macro is used both to check and to write tape labels. A LABELDEF command must be supplied separately to use the macro. The tape must be positioned correctly (at the label to be checked or at the place where label is to be written) before issuing the macro. TAPECTL may be used to position the tape. TAPESL reads or writes only one tape record unless SPACE=YES is specified. The format of the TAPESL macro is:

[label]	TAPESL	function[,device],LABID=labeldefid[,MODE=mode] [,BLKCNT=blkcnt][,ERROR=erraddr] [,SPACE={YES/NO}][,TM={YES/NO}]

where:

label
is an optional statement label.

function
is one of the following:

HIN	checks input HDR1 label.
HOUT	writes HDR1 label.
EIN	checks input EOF1 label.
EOUT	writes output EOF1 label.
EVOUT	writes output EOV1 label.

device
is one of the following:

TAP*n*
cuu
indicates the symbolic tape identification (TAPn) or the virtual device address (cuu) of the tape. If omitted, 181 is assumed. The following symbolic names and virtual devices are supported:

Symbolic Name	Virtual Address	Symbolic Name	Virtual Address
TAP0	180	TAP8	288
TAP1	181	TAP9	289
TAP2	182	TAPA	28A
TAP3	183	TAPB	28B
TAP4	184	TAPC	28C
TAP5	185	TAPD	28D
TAP6	186	TAPE	28E
TAP7	187	TAPF	28F

MODE=mode
specifies the number of tracks, density, and tape recording technique options. It must be in the following form:

```
([track],[density],[trtch])
```

track

7 indicates a 7-track tape (implies density=800 and trtch=O).

9 indicates a 9-track tape (implies density=800).

18 indicates a 18-track tape (implies density=38K).

density

200, 556, or 800 for a 7-track tape.

800, 1600, or 6250 for a 9-track tape.

38K for an 18-track tape.

trtch

indicates the tape recording technique for 7-track tape. One of the following must be specified:

O	odd parity, converter off, translator off.
OC	odd parity, converter on, translator off.
OT	odd parity, converter off, translator on.
E	even parity, converter off, translator off.
ET	even parity, converter off, translator on.

LABID=labeldefid
specifies the 1- to 8-character name on the LABELDEF command to be used for the file. (A separate LABELDEF statement must be specified for the file before the program containing TAPESL is executed.)

BLKCNT=blkcnt
specifies the block count to be inserted in an EOF1 or EOV1 label on output or used to check against on input. This field is only used for functions EOUT, EIN, or EVOUT. If not specified, the output block count is set to 0. This field may also be specified as a register number enclosed within parentheses when a general register contains the block count.

ERROR=erraddr
specifies the address of an error routine to be given control if an error of any kind occurs during label processing. If ERROR= is not coded and an error occurs, control is returned to the next sequential instruction in the calling program. If you request the EIN function and a block count error is detected, control is transferred to your error routine if you specify an ERROR= parameter that contains an address different from the next sequential instruction. If no error return is specified or the ERROR= address is the same as the normal return, a block count error causes message 425R to be issued.

SPACE=YES
NO
may be specified for functions HIN and EIN. If YES is specified, the tape is spaced, after processing, beyond the tapemark at the end of the label record. If NO is specified, the tape is not moved after the label has been processed. YES is the default.

TM=YES
NO
may be specified for functions HOUT, EOUT, and EVOUT. If YES is specified, a single tapemark is written after a HDR1 or EOV1 label. Two tapemarks are written after an EOF1 label. If NO is specified, no tapemarks are written. YES is the default.

Usage Notes:

1. If the MODE option is not specified, the tape drive will use the default density of the drive. The default density for dual density drives is the highest density. This is true even if the TAPE MODESET command is entered right before the macro is issued.

2. The input functions HIN and EIN read a tape label and check to see if it is the type specified. They also check any fields in the tape label that have been specified explicitly (no default) in the LABELDEF statement (indicated by LABID). Any discrepancies between the fields in the LABELDEF statement and the fields on the tape label cause an error message to be issued and an error return to be made.

3. The output functions HOUT, EOUT, and EVOUT write a tape label of the requested type on the specified tape. The values of fields within the labels are those specified or defaulted to in the LABELDEF command. See the description of the LABELDEF command in *VM/SP CMS Command Reference* for information about the default fields.

4. For a more complete discussion of tape label processing, see "CMS Tape Label Processing" in the *VM/SP CMS User's Guide*.

Error Conditions:

If an error occurs, register 15 contains one of the following error codes:

Code	Meaning
24	Invalid device type specified or invalid device attached.
28	LABELDEF cannot be found.
32	Error in checking tape label or block count error.
36	Output tape is file-protected.
40	End of file or end of tape occurred.
100	Tape I/O error occurred.

TRANTBL

Use the TRANTBL macro instruction to generate a DSECT for the system character set translation tables.

The format of the TRANTBL macro instruction is:

[label]	TRANTBL

where:

label
is an optional label for the statement. The first statement in TRANTBL macro expansion is labeled TRANTBL.

Usage Notes:

1. The TRANTBL macro instruction expands as follows:

```
TRANTBL  DSECT
TRANSTD  DS    CL256      Standard uppercase table
TRAST77  DS    CL256      EBCDIC -> 3277 Character Set
TRAST78  DS    CL256      EBCDIC -> 3278 Character Set
TRAAPL77 DS    CL256      EBCDIC -> 3277 APL Character Set
TRAAPL78 DS    CL256      EBCDIC -> 3278 APL Character Set
TRATXT77 DS    CL256      EBCDIC -> 3277 Text Character Set
TRATXT78 DS    CL256      EBCDIC -> 3278 Text Character Set
TRAPL7EC DS    CL256      EBCDIC -> 3277/APL Compound Chars
TRAPL7CE DS    CL256      3277/APL Compound Chars -> EBCDIC
TRAPL8EC DS    CL256      EBCDIC -> 3278/APL Compound Chars
TRAPL8CE DS    CL256      3278/APL Compound Chars -> EBCDIC
TRATX7EC DS    CL256      EBCDIC -> 3277/Text Compound Char
TRATX7CE DS    CL256      3277/Text Compound Char -> EBCDIC
TRATX8EC DS    CL256      EBCDIC -> 3278/Text Compound Char
TRATX8CE DS    CL256      3278/Text Compound Char -> EBCDIC
TRATX7ES DS    CL256      EBCDIC -> 3277/Text Single Char
TRATX7SE DS    CL256      3277/Text Single Char -> EBCDIC
```

WAITD

Use the WAITD macro instruction to cause the program to wait until the next interruption occurs on the specified device. The format of the WAITD macro instruction is:

[label]	WAITD	device...[,devicen] [,ERROR=erraddr]

where:

label
is an optional statement label.

devicen
specifies the device(s) to be waited for. One of the following may be specified:

symn
indicates the symbolic device name and number, where:

sym
is CON, DSK, PRT, PUN, RDR, or TAP.

n
indicates a device number.

user
is a four-character symbolic name specified a HNDINT macro issued for the same device.

ERROR=*erraddr*
specifies the address of an error routine to be given control if an error is found. If ERROR= is not coded and an error occurs, control returns to the next sequential instruction in the calling program, as it does if no error occurs.

Usage Notes:

1. Use the WAITD macro instruction to ensure completion of an I/O operation. If an interruption has been received and not processed from a device specified in the WAITD macro instruction, the interruption is processed before program execution continues.

2. When the interruption has been completely processed, control is returned to the caller with the name of the interrupting device in register 1.

3. If an HNDINT macro instruction issued for the same device specified ASAP and an interruption has already been processed for the device, the wait condition is satisfied.

4. If an HNDINT macro instruction issued for the same device specified WAIT and an interruption for the device has been received, the interruption handling routine is given control.

5. The interruption routine determines if an interruption is considered processed or if more interruptions are necessary to satisfy the wait condition. For additional information see the discussion of the HNDINT macro instruction.

Error Conditions:

If an error occurs, register 15 contains a 1 to indicate that an invalid device number was specified.

WAITECB

Use the WAITECB macro instruction to wait on an ECB (Event Control Block) or a list of ECBs. The WAITECB macro suspends processing until a specific ECB or the ECBs in a list have been posted. "Posting" the ECB involves turning the "event complete" bit on. Event completion is signaled by setting the appropriate completion bit (based on the ECB format) in an asynchronous routine such as a timer or interrupt handler exit.

The four formats of the WAITECB macro instruction are:

- Standard
- List (MF=L)
- Complex List (MF=(L,addr[,label]))
- Execute (MF=(E,addr))

Standard Format

The Standard Format of the WAITECB macro instruction is:

[label]	WAITECB	{ECB=addr [,FORMAT=OS|VSE] {[count] ,ECBLIST=addr [,FORMAT=OS|VSE]}

where:

label
is an optional statement label.

count
specifies the number of ECBs to be posted before returning to the caller. It is specified as a decimal number or a register (2-12), enclosed in parentheses. If not specified, the value 1 is assumed.

The count operand is only valid with the ECBLIST form of the macro. If it is specified with the ECB form of the macro, an MNOTE error message is issued.

addr
is an assembler program label or an address stored in a general register (2-12), enclosed in parentheses.

addr is the address of an ECB on a fullword boundary or the address of a virtual storage area containing one or more consecutive fullwords on a fullword boundary. Each fullword contains the address of an ECB. The end-of-list indicator has two forms:

1. If FORMAT=OS is specified (or defaulted), the high order bit (bit 0) in the last fullword must be set to 1.

2. If FORMAT=VSE is specified, the byte following the last fullword of the list must be non-zero.

FORMAT
specifies the format of the ECB(s) used. If OS is specified, the bit tested for "event completed" is byte 0 bit 1. If VSE is specified, the bit tested is byte 2 bit 0. If this parameter is omitted, it defaults to FORMAT=OS. With ECBLIST, ECB formats cannot be mixed.

List Format

When MF=L is coded, the WAITECB macro has the following format:

[label]	WAITECB MF=L	[,ECB=label][,FORMAT=OS|VSE] [,count][,ECBLIST=label][,FORMAT=OS|VSE]

The parameters have the same meaning as in the Standard Format except for the following:

MF=L
indicates that the parameter list is created in-line. No executable code is generated. Register notation cannot be used for macro parameter addresses.

Note: When you use the MF= parameter, all other parameters are optional. Before the function is executed, a valid combination of parameters must be specified by the List and Execute Formats of the macro.

Complex List Format

When MF=(L,*addr*[,*label*]) is coded, the WAITECB macro has the following format:

[label]	WAITECB MF=(L,addr[,label])	[,ECB=addr][,FORMAT=OS|VSE] [,count][,ECBLIST=addr] [,FORMAT=OS|VSE]

The parameters have the same meaning as in the Standard Format except for the following:

MF=(L,addr[,label])
indicates that the parameter list is created in the area specified by addr. The address may represent an area within your program or an area of free storage obtained by a system service. You can determine the size of the parameter list by coding the label operand. The macro expansion equates label to the size of the parameter list. This format of the macro produces executable code to move the data into the

parameter list specified by addr. However, it does not generate the instructions to invoke the function. If this version of the List Format is used, it must be executed before any related invocation of the Execute Format.

count
If the count operand is not specified, no default is assumed so as not to override a count that may have been specified by the List Format of the WAITECB macro.

Note: When you use the MF= parameter, all other parameters are optional. Before the function is executed, a valid combination of parameters must be specified by the List and Execute Formats of the macro.

Execute Format

When MF=(E,addr) is coded, the WAITECB macro has the following format:

[label]	WAITECB MF=(E,addr)	[,ECB=addr][,FORMAT=OS|VSE] [,count][,ECBLIST=addr][,FORMAT=OS|VSE]

The parameters have the same meaning as in the Standard Format except for the following:

MF=(E,*addr*)
indicates that instructions are generated to execute the WAITECB function. addr represents the location of the parameter list. Information in the parameter list may be changed by specifying the appropriate operands on the macro.

count
If the count operand is not specified, no default is assumed so as not to override a count that may have been specified by the List Format of the WAITECB macro.

Note: When you use the MF= parameter, all other parameters are optional. Before the function is executed, a valid combination of parameters must be specified by the List and Execute Formats of the macro.

ECB Formats

ECBs are standard mechanisms used to synchronize multiple events. The process of turning on the "event complete" bit is referred to as "posting" the ECB. ECBs are fullwords and have the following OS or VSE format:

OS Format:

```
bit 0      WAIT bit
bit 1      Event completed bit
bit 2-7    Completion code
bit 8-31   Unused
```

VSE Format:

```
byte 0-1     Reserved
byte 2
   bit 0     Traffic bit (Event completed bit)
   bit 1-7   Reserved
byte 3       Reserved
```

Note: For an OS format ECB, the OS simulation POST macro is an acceptable alternative to turning the event completed bit on yourself.

WaitConsole I/O Wait

The CMS nucleus, DMSNUC, contains an ECB that makes it easier to wait for a console I/O in a series of multiple events. This ECB is called CON1ECB and is defined in DEVTAB. This ECB has two "event completed" bits. The format of CON1ECB is:

```
byte 0
   bit 0     Wait bit
   bit 1     Event completed bit number 1
   bit 2-7   Completion code
byte 1       Reserved
byte 2
   bit 0     Event completed bit number 2
   bit 1-7   Reserved
byte 3       Reserved
```

When the terminal input buffer contains a line, both event completed bits in the CON1ECB ECB are posted.

Usage Notes:

1. The number of ECBs specified in the macro are sequentially searched to check if they have been "posted."

2. If the user specified a count larger than the number of ECBs in the ECBLIST, execution of this macro results in a permanent wait.

3. If the count is negative or zero, this function results in a NOP. (NOP means no-operation instruction; the program just proceeds to the next instruction.)

4. The console ECB only applies to the terminal input buffer. No ECB is associated with the program stack.

WAITT

Use the WAITT macro instruction to cause the program to wait until all of the pending terminal I/O is complete.

The format of the WAITT macro instruction is:

[label]	WAITT	

where:

label
is an optional statement label.

Usage Note:

1. The WAITT macro instruction synchronizes input and output to the terminal; it ensures that the console stack is cleared before the program continues execution. Also, you can ensure that a read or write operation is finished before you modify an I/O buffer.

WRTAPE

Use the WRTAPE macro instruction to write a record on the specified tape drive. The format of the WRTAPE macro instruction is:

[label]	WRTAPE	buffer,length [,device] [,MODE=mode] [,ERROR=erraddr] [,TRAN=tran]

where:

label
is an optional statement label.

buffer
specifies the address of the record to be written. It may be:

lineaddr
the symbolic address of the line.

(reg)
a register containing the address of the buffer.

length
specifies the length of the line to be written. It may be specified in either of two ways:

n
a self-defining term indicating the length.

(reg)
a register containing the length.

device
specifies the device to which the record is to be written. If omitted, TAP1 (virtual address 181) is assumed. It may be:

TAP*n*
cuu
indicates the symbolic tape identification (TAPn) or the virtual device address (cuu) of the tape you are writing to. The following symbolic names and virtual devices are supported:

Symbolic Name	Virtual Address	Symbolic Name	Virtual Address
TAP0	180	TAP8	288
TAP1	181	TAP9	289
TAP2	182	TAPA	28A
TAP3	183	TAPB	28B
TAP4	184	TAPC	28C
TAP5	185	TAPD	28D
TAP6	186	TAPE	28E
TAP7	187	TAPF	28F

MODE = *mode*
specifies the number of tracks, density, and tape recording technique. It must be in the following form:

```
([track],[density],[trtch])
```

track

7 indicates a 7-track tape (implies density = 800 and trtch = 0).

9 indicates a 9-track tape (implies density = 800).

18 indicates a 18-track tape (implies density = 38K).

density

200, 556, or 800 for a 7-track tape

800, 1600, or 6250 for a 9-track tape.

38K for an 18-track tape.

trtch

indicates the tape recording technique for 7-track tape. One of the following must be specified:

O	odd parity, converter off, translator off.
OC	odd parity, converter on, translator off.
OT	odd parity, converter off, translator on.
E	even parity, converter off, translator off.
ET	even parity, converter off, translator on.

ERROR = *erraddr*
specifies the address of an error routine to be given control if an error is found. If ERROR= is not coded and an error occurs, control returns to the next sequential instruction in the calling program, as it does if no error occurs.

TRAN=*tran*

specifies the tape write mode available only for the 3480 Magnetic Tape Subsystem. The two write modes are BUFFered and IMMEDiate. The default is the BUFF Mode. Use of the IMMED mode causes a severe performance degradation.

BUFF

specifies that data is to be transferred from the processor to the tape control unit. As soon as the data transfer is complete, a "complete" signal is issued, and the processor and tape control unit are disconnected from each other. The tape control unit then writes the data on tape and performs error checking and recovery procedures.

IMMED

specifies that data is to be physically written on tape and "read-back" checked (verified) by the microprogram in the tape control unit. A "complete" signal is issued and the processor and tape control unit are disconnected from each other after the data is actually on the tape.

Usage Notes:

1. If the MODE option is not specified, the tape drive will use the default density of the drive. The default density for dual density drives is the highest density. This is true even if the TAPE MODESET command is entered right before the macro is issued.
2. You need not specify the MODE option when you are writing to a 9-track tape and want to use the default density, nor when you are writing to a 7-track tape with a density of 800 bpi, odd parity, with data converter and translator off.
3. You must specify the MODE option (track and density) when writing on a 3480 tape.
4. The BUFF option (the default) of the TRAN parameter for the 3480, signals CHANNEL END and DEVICE END when data has been completely transmitted to the channel. To prevent a delayed unit check (that occurs when the data is not correctly written to the tape cartridge), you may issue a tape control function such as a WTM or REW. Tape control functions can be issued with the TAPECTL macro.

Error Conditions: If an error occurs, register 15 contains one of the following error codes:

Code	Meaning
1	Invalid function or parameter list
2	End of file or end of tape
3	Permanent I/O error
4	Invalid device identification
5	Tape not attached
6	Tape is file-protected
7	Invalid device attached

WRTERM

Use the WRTERM macro instruction to display a line at the terminal. The format of the WRTERM macro instruction is:

[label]	WRTERM	line [,length] [,EDIT=code] [,COLOR=color]

where:

label
is an optional statement label.

line
specifies the line to be displayed. It may be one of three forms:

linetext
the actual text line enclosed in single quotation marks.

lineaddr
the label on the statement containing the line.

(reg)
a register containing the address of the line.

length
specified the length of the line. If the line is specified within quotation marks in the macro instruction, the length operand may be omitted. The length may be specified in either of two ways:

n
a self-defining term indicating the length.

(reg)
a register containing the length.

EDIT=*code*
specifies whether the line is to be edited:

YES
indicates that trailing blanks are to be removed and a carriage return added to the end of the line. YES is the default value.

NO
indicates that trailing blanks are not to be removed and no carriage return is to be added.

LONG
indicates the line may exceed 130 bytes. No editing is performed.

COLOR=*color*
indicates the color in which the line is to be typed, if the typewriter terminal has a two-color ribbon:

B
indicates that the line is to be typed in black. This is the default.

R
indicates that the line is to be typed in red.

Usage Notes:

1. The maximum line length is 130 characters for a black line and 126 characters for a red line.

2. If EDIT=LONG, COLOR must be specified as "B." In this case, you may write as many as 1760 bytes with a single WRTERM macro instruction. You are responsible for embedding the proper terminal control characters in the data. (This operand is for use primarily with VS APL programs.)

3. You may want to use the WAITT macro instruction to ensure that terminal I/O is complete before continuing program execution.

4. When EDIT=NO or LONG is used, the same output to graphics devices and to line terminal devices may appear inconsistent because of differences in device characteristics.

5. Use the LINEWRT macro instead of the WRTERM macro for increased flexibility. The LINEWRT macro supports all of the functions that WRTERM does, and it also allows the user to specify features such such as a virtual screen name, color, and extended highlighting. (See the LINEWRT macro.)

Chapter 2. CMS Functions

This part describes functions that are available to the CMS user.

Execute CMS functions from application programs by setting up a parameter list and then issuing an SVC 202. When you want to execute a function in your program, load the address of the function parameter list into Register 1 and issue the SVC 202 as follows:

```
LA    1,Parameter List
SVC   202
DC    AL4(ERROR)
```

where:

ERROR is a routine to handle nonzero return codes returned in register 15 after execution of the SVC call.

If you want to ignore errors, you can code the sequence:

```
LA    1,Parameter List
SVC   202
DC    AL4(1)
```

If the function completes normally, this sequence causes execution of the next sequential instruction. However, if an error occurs while executing the function and the program requires successful execution of the function, abnormal termination of your program may result.

The following CMS functions are described in this section:

- ATTN
- DISKID
- LANGADD
- LANGFIND
- NUCEXT
- TODACCNT
- WAITRD

ATTN

Use the ATTN function to insert a line of input into the program stack. ATTN may be executed from an assembler language program via SVC 202 with the following parameter list:

```
PLIST    DS   OD
         DC   CL8'ATTN'
         DC   CL4'order'      where order may be LIFO or FIFO.
*                             FIFO is the default
         DC   AL1(length)     length of line to be stacked
         DC   AL3(addr)       address of line to be stacked
```

Usage Notes:

1. The line that ATTN stacks is extracted from the program stack when WAITRD is executed to read a line of input. (See the WAITRD function description for details of WAITRD function.)

2. ATTN stacks lines of up to 255 characters.

Responses:

None

Return Codes:

Code	Meaning
0	Function successfully completed
25	No more storage

DISKID

Use the DISKID function to obtain information on the physical organization of a RESERVEd minidisk. The DISKID function obtains the virtual address, the block size, and the offset of the minidisk.

A disk does not have to be accessed when you use the DISKID function. If the disk is accessed, the DISKID function obtains the necessary information from the Active Disk Table (ADT) for that disk. If the disk is not accessed, CMS obtains the information from the disk's label.

You need this information in order to use the CP DASD Block I/O System Service with a CMS formatted minidisk that was RESERVEd.

DISKID is executed from an assembler language program via SVC 202 with the following parameter list:

```
PLIST    DS  0D
         DC  CL8'DISKID'
         DC  CL8'ddname'     ddname for the mini-disk
         DS  XL2             Virtual address
         DS  H               Blocksize
         DS  F               Offset
         DS  D               Reserved
```

Usage Notes:

1. The parameter list must begin on a doubleword boundary.

2. The program calling the DISKID function fills the first two doublewords of the parameter list.

3. The second doubleword contains the "ddname" specified in the FILEDEF command issued for the Block I/O disk.

4. The third doubleword is filled upon completion of the DISKID function. It contains:

   - the virtual address of the mini-disk for which a "ddname" exists
   - the block size of the mini-disk
   - the offset of the mini-disk. This offset associates a physical block number to the first data block of the unique file on disk that was previously RESERVEd. The block number represents the number of sequential blocks used on the disk by the CMS file system to implement its structure. Data block number one for the file is then physical block number one plus offset.

Return Codes:

Register 15 contains one of the following codes:

Code	Meaning
0	Return information supplied in parameter list
4	High order byte of register 1 is not zero
12	DASD not reserved with the RESERVE command
28	"ddname" not defined or "ddname" not filedef'ed to DISK cuu
100	Disk not attached
1xx	An I/O error occurred; xx = the return code from DIAG 18 (CKD devices)
2xx	An I/O error occurred; xx = the return code from DIAG 20 (FBA devices)

LANGADD

Use the LANGADD function to add a LANGBLK to the language block chain.

LANGADD does this by:

1. making a copy of the LANGBLK pointed to in the plist, and then
2. adding it to the language block chain of LANGBLKs (if a LANGBLK for the application is not already on the chain).

This allows an application to have one language as part of its "nucleus," just as CMS does.

You can execute LANGADD from a program via an SVC 202 with the following parameter list:

```
DS  OF
DC  CL8'LANGADD'
DS  A(addr of LANGBLK)
DC  8X'FF'
```

Usage Note:

The SET LANGUAGE command will not be able to restore the information in the LANGBLK if another language is set and the original language is restored. The application must request LANGADD to add the LANGBLK again.

Return Codes:

Code	Meaning
0	The function was successfully completed.
4	A LANGBLK for the application is already on the language block chain.

LANGFIND

Use the LANGFIND function to get the address of an application's language control block.

Each application may have a language control block (LANGBLK) which contains pointers to all language-related information. LANGFIND allows you to locate the LANGBLK for a specific application.

You can execute LANGFIND from a program via an SVC 202 with the following parameter list:

```
DS  OF
DC  CL8'LANGFIND'
DS  CL4'application id'
DS  A(addr of LANGBLK)
DC  8X'FF'
```

Upon return, the four bytes following the application id will either contain:

1. the address of the LANGBLK, if the application id requested was found, or
2. zero, if no LANGBLK contained the application id that was requested.

NUCEXT

The nucleus extension function (NUCEXT) allows you to identify command entry points in programs established in free storage, so that they may be called by a SVC 202 as if they were nucleus commands. They thus become nucleus extensions. You can also create your own Immediate commands with the NUCEXT function.

NUCEXT builds a chain of SCBLOCKS in storage for nucleus extensions. The chain of nucleus extensions is reordered each time a command is found on the chain. The reordering puts the most frequently used commands at the beginning of the chain.

NUCEXT is a name given to a group of commands that all make use of an internal function named NUCEXT. The actual commands provided for manipulation of nucleus extensions are:

NUCXLOAD Loads an ADCON-free or relocatable module into free storage and installs it as a nucleus extension.

NUCXDROP Cancels a nucleus extension and releases the corresponding storage.

NUCXMAP Prints on the console or stacks a list of the nucleus extensions.

Use NUCEXT to access user-written programs without having to do disk read operations (as would be required for modules) or to avoid thrashing in the transient or user areas when several programs are used repeatedly (the same programs are loaded many times).

Use NUCEXT for gathering statistics, filtering commands for various purposes, creating anchors for data kept in free storage until the next CMS IPL, and special operations during CMS abnormal end processing.

Nucleus extensions with the IMMCMD option can receive control as user-defined Immediate commands or as regular commands. Nucleus extensions with the ENDCMD option, receive control at normal end-of-command processing. The ENDCMD nucleus extensions only receive control after a command is entered from the virtual console. They do not receive control if the command was issued from an EXEC, a user program, or CMS SUBSET mode. Unlike transient routines or user programs, nucleus extensions are retained until they are unloaded explicitly, or as a side effect of abnormal end cleanup for those using free storage of type 'user' (which is reclaimed during an abnormal end) or which are not designated as system routines to survive abnormal end. Nucleus extensions can have the same name as existing CMS nucleus commands or functions. If they do have the same name, the extensions override the existing nucleus commands or functions. Only nucleus functions invoked via SVC 202 can

be overridden. However, two existing nucleus functions, RDBUF and WRBUF, cannot be overridden. It is possible to create a nucleus extension that can call another nucleus extension having the same name. This allows a nucleus extension to "frontend" another nucleus extension. The techniques necessary to perform this call are complex and require assembler language programming. This override process may not be possible in all cases.

The nucleus extension that was the last nucleus extension to be established receives control first. This is the first nucleus extension on the SCBLOCK chain with a name that matches the requested name.

The nucleus extension may perform whatever processing that it requires. To pass control to another nucleus extension having the same name, it must first change the name field of the original SCBLOCK to a unique name.

The original nucleus extension may now issue an SVC 202 for the nucleus extension control that is to be passed. The original nucleus extension can restore the original contents of general registers 0 and 1 before this call.

Control is passed to the next nucleus extension with the same name on the SCBLOCK chain. The PLIST that the nucleus extension receives is the PLIST that was pointed to by registers 0 and 1 when the SVC 202 was issued on the first nucleus extension.

On return from the second nucleus extension, the original nucleus extension must now issue an SVC 202 for itself. The name used for this SVC 202 must be the unique name that was placed in the SCBLOCK earlier. This call reorders the SCBLOCK chain, placing the original nucleus extension at the head of the SCBLOCK chain. The nucleus extension must be designed to recognize these special reorder calls. Reorder calls can be determined by checking the parameter list that is pointed to by register 1 upon entry. If the unique name is the first token in the PLIST, then this is a reorder call. Control should only be returned to the caller; typically, no processing should be performed.

The original nucleus extension should now restore the name field of its SCBLOCK to its original name from the unique name. Control may now be returned to the original caller.

Nucleus Extensions and Abnormal Ends

Types of Nucleus Extensions

There are two types of nucleus extensions, "system" and "user," differentiated by their behavior during a CMS abnormal end. The former will survive an abnormal termination of a user program (abnormal end), whereas the latter will not.

Note: Because CMS reclaims all storage of type "user" during the abnormal end cleanup phase, any nucleus extension in "user" storage is deleted during abnormal end, regardless of its "system" attribute. The storage obtained for "user" type nucleus extensions code must be be doubleword aligned to the next doubleword or DMSFRE errors will occur during ABEND processing.

Because of this storage reclamation during abnormal end, programs which build data structures in free storage of type 'user' but keep pointers in storage of type 'system' need to know when abnormal end cleanup occurs (e.g., after HX).

Service Calls: PURGE and RESET

A program's need to know about abnormal end cleanup is supported by the idea of a service call. When a nucleus extension is declared (via NUCEXT), it may request that it receive a service call under appropriate circumstances. There are two standard service calls supported by NUCEXT. The PURGE service call is issued during CMS abnormal end cleanup. The RESET service call is issued by the NUCXDROP program when a nucleus extension is explicitly unloaded. The service calls allow programs with several entry points to cancel these at the same time, or to free storage areas.

Note: A note on service calls during an abnormal end: Do not stack during a service call. This causes the system to allocate storage that is not accounted for during abnormal end. The sequence of events that occur during an abend are documented in *VM/SP CMS for System Programming*.

The SYSTEM and SERVICE Attributes:

Nucleus extensions may or may not have the "SYSTEM" attribute and/or the "SERVICE" attribute. These attributes determine the handling of a nucleus extension during abnormal end processing.

If a nucleus extension has the "SYSTEM" attribute, it remains active after an abnormal end. It is your responsibility to see that such a nucleus extension is loaded into nucleus storage, not user storage (which is recovered after an abnormal end).

If a nucleus extension has the "SERVICE" attribute, it is called during abnormal end processing with the parameter list:

```
DS      0F
DS      CL8'nucleus extension name'
DC      CL8'PURGE'
DC      8X'FF'
```

The high order byte in register 1 is set to X'FF'. A nucleus extension may have the "SYSTEM" and "SERVICE" attributes in any combination.

Nucleus Storage:

During abnormal end recovery, nucleus storage used by nucleus extensions behaves as follows:

1. When a nucleus extension has the "SYSTEM" attribute, it should be in nucleus storage and the length word is used by abnormal end recovery to account for the amount of storage used by that program.

2. If a nucleus extension does not have the "SYSTEM" attribute but is in nucleus storage anyway, that storage will be recovered during abnormal end.

When a nucleus extension obtains nucleus-type free storage other than what is accounted for by the origin and length fields in the SCBLOCK, it should either:

1. Use the "SERVICE" flag so that it is called with the PURGE parameter list during abnormal end, at which time it returns any nucleus-type storage it obtained (but not that described in its SCBLOCK).

2. If it has the "SYSTEM" attribute, account for any extra nucleus storage which is to be kept through an abnormal end by adding the length in doublewords of such storage into the NUCXFRES field in NUCON. It's a good idea to update this field as soon as the storage is obtained. This is required if the nucleus extension does not have the "SERVICE" attribute.

Nucleus extensions remain in effect until cancelled, either explicitly or implicitly. Implicit cancellation normally occurs only for nucleus extensions of the "user" type (during an abnormal end cleanup time when all storage of "user" type is reclaimed). Explicit cancellation does not release the storage (if any) occupied by the nucleus extension: that is the responsibility of the program that issues the cancellation (usually the program NUCXDROP).

Using the NUCEXT function affects the command resolution strategy of DMSITS when a SVC 202 is processed. Nucleus extensions are sought before functions in the real CMS nucleus (i.e., one which is defined by an entry in DMSFNC). This gives the user the ability to intercept, filter, augment, etc., the "real" nucleus functions.

ENDCMD Attribute

A nucleus extension with the ENDCMD option receives control at normal end-of-command processing. At end-of-command processing the CMS console handler invokes all nucleus extensions having the ENDCMD option. The nucleus extensions are invoked by an SVC 202. Register 1 points to the PLIST, the high order byte of register 1 is set to X'FE' to indicate an end-of-command call. The PLIST used to invoke an ENDCMD extension is:

```
DS   0F
DS   CL8'nucleus extension name'
DC   CL8'ENDCMD'
DS   F'return code'
DC   8X'FF'
```

where:

"return code" is the value returned to CMS in register 15 by the terminating command.

Linkage Conventions

When a nucleus extension is declared, the following information must be provided:

- The NAME of the command implemented by the nucleus extension.
- The PSW to be used when passing control to the nucleus extension.
- The address and length of the storage area occupied by the program. The length must be rounded up to doubleword alignment.
- Flag bits to indicate either type "user" or "system," and indicate whether service calls are desired.
- Flag bits should be used to indicate if the ENDCMD or IMMCMD options are desired.

Secondary entry points are declared by indicating a storage size of zero. The PSW specifies the system mask, the PSW key to be used, the program mask (and initial condition code), and the starting address for execution. The problem-state bit and machine-check bit may be set. The machine-check bit has no effect in CMS under CP. The EC-mode bit and the wait-state bit cannot be set (they are always forced to zero). The flag bits are encoded in the third byte of the PSW. Also, one byte of user defined flags and one 4-byte user-defined word can be associated with the nucleus extension, and referred to when the entry point is subsequently called.

Entry into a Nucleus Extension: On entry to a nucleus extension, the register contents are:

```
R0          Address of extended parameter list  (if
                 one was provided by the caller).
R1          Address of the command name  (and the
                 tokenized parameter list).
R2          Address of SCBLOCK with NUCEXT extension.
R12         Entry point address.
R13         24-word save area address.
R14         Return address (CMSRET).
R15         Entry point address.
```

This is the standard entry point convention except that R2 points to the SCBLOCK.

NUCEXT

The NUCEXT function queries, declares, or cancels user nucleus extensions. NUCEXT can be issued as a command only for its query function. With one argument, 'name,' it returns either:

```
0    'name' is a user nucleus extension (found it).
```

or

```
1    'name' not found.
```

PLISTs

As a function (called from a program), NUCEXT takes the following PLIST:

Declare PLIST:

```
NUCX     DS  0F                 PLIST TO DECLARE NUCLEUS EXTENSION
         DC  CL8'NUCEXT'
NUCXNAME DC  CL8'name'          COMMAND NAME
NUCXPSW  DC  XL2'0000',AL2(0)   SYSTEM MASK, STORAGE KEY, ETC
NUCXADDR DC  A(*-*)             ENTRY ADDRESS, -1 for QUERY
         DC  A(0)               USER WORD
NUCXORG  DC  A(*-*)             LOAD ADDRESS
NUCXLEN  DC  A(*-*)             SIZE, IN BYTES ROUNDED TO THE NEXT
                                DOUBLEWORD.
```

This declares 'name' as a nucleus extension and puts an SCBLOCK at the head of the NUCEXT chain. The name may already be defined, in which case the previous declaration will be hidden until the present one is cancelled. Return code 25 means not enough storage was available to allocate the necessary SCBLOCK.

The third and fourth bytes of the start-up PSW (interrupt code) are used as flag bytes. The format of the PSW is:

```
AL1(system mask)          (EC-mode bit forced to 0)
AL.4(storage key)
BL.4'0MWP'
AL1(NUCEXT flags)         System=X'80',
                          Service=X'40'
                          End of command=X'10'
                          Immediate=X'04'
AL1(user flag)            May be used for private purpose.
A(entry point)
```

Cancel PLIST:

```
CL8'NUCEXT'
CL8'name'
XL4'irrelevant'
A(0) identifies the cancel function
```

This cancels the nucleus extension or gives a return code of 1 if 'name' is not found. The storage occupied by the program calling for this nucleus extension is not freed. This is the responsibility of the cancelling program.

Query PLIST:

```
CL8'NUCEXT'
CL8'name'
XL4'irrelevant'          Receives A(SCBLOCK).
XL4'FFFFFFFF'            identifies the query function
```

This form returns the address of the SCBLOCK if 'name' is found, otherwise it changes nothing and gives a return code of 1.

Note that if 'NUCEXT name' is called from a command level or from an EXEC file, the Query PLIST is the form of PLIST which will be issued.

Get Anchor PLIST:

```
CL8'NUCEXT'
CL8'irrelevant'
A(*-*)               Receives value (not address)
                       of NUCEXT list anchor or 0 if
                       there are no nucleus extensions.
A(1)                 Indicates request for anchor.
```

Nucleus Extensions as Immediate Commands

When a nucleus extension is established with the IMMCMD option, it can be invoked as a regular command or as an Immediate command. In addition to having an SCBLOCK, a nucleus extension with IMMCMD attribute also has a similar control block, called an IMMBLOK, associated with it.

Nucleus extensions with the IMMCMD attribute are entered as Immediate commands when they are invoked by the CMS console interrupt handler. This occurs when a particular command that has been established as an Immediate command is entered by the terminal user while CMS is busy.

Nucleus extensions with the IMMCMD attribute can be overridden by an identically named nucleus extension (for example, NUCXLOAD with the PUSH option). If the new nucleus extension does not have the IMMCMD attribute but does have the same name as an existing nucleus extension with the IMMCMD attribute, the nucleus extension with the IMMCMD attribute is disabled as an Immediate command.

Entry conditions to a nucleus extension as an Immediate command are identical to the entry conditions that occur when a nucleus extension is invoked via SVC 202, except for the following conditions:

- The high order byte of register 1 contains X'06'. This indicates that the nucleus extension was invoked as an Immediate command. When invoked via SVC 202, the high order byte of register 1 is normally X'01' or X'0B'.
- Register 2 contains the address of an IMMBLOK.
- Register 14 contains the return address that is located in the CMS console interrupt handler (DMSCIT).

With respect to common information (for example, command name and user word), displacements within the IMMBLOK are identical to those in an SCBLOCK). These displacements are as follows:

Displacement	Offset Information
0	Pointer to next IMMBLOK
4	User word
8	Command name
20	Entry point address

For a general discussion of Immediate commands in CMS, see *VM/SP CMS for System Programming*.

TODACCNT

Use the TODACCNT function to issue a DIAGNOSE 70 for activating the time-of-day clock accounting interface. Using the TODACCNT function helps to avoid DIAGNOSE 70 calls (a specification exception).

The TODACCNT function has two subfunctions, ENABLE and QUERY.

- ENABLE tells CMS to issue a DIAGNOSE 70 instruction to indicate to CP that the virtual machine wishes to receive timing information. Each time the virtual machine is dispatched, CP provides the accumulated processor time the virtual machine has used and the time-of-day that the virtual machine was dispatched. This information is stored in a 16-byte area in page zero.

- QUERY function returns the 16 bytes of timing information supplied by CP as a result of the enable function.

TODACCNT is executed from a program via SVC 202 with the following parameter list:

```
PLIST    DS   OD
         DC   CL8'TODACCNT'
         DC   CL8'function'     function is ENABLE or QUERY.
*                               ENABLE or QUERY should be an
*                               8-byte field.
         DC   Bx(addrùfield)    for ENABLE, a 4-byte address
                                field where the timing
*                               information is to be stored
*                               (provided the return code is
*                               0 or 4).  For QUERY, a 16-byte
*                               field where CMS will return
*                               the 16-bytes of timing
*                               information that is transferred
*                               transferred from page zero
*                               (provided the return code is 0).
```

Usage Notes:

1. The parameter list must be on a double-word boundary.

2. AN error return address must be supplied in the 4 bytes immediately following the SVC 202 instruction. If the return code (register 15) contains a nonzero value after returning from the SVC call, control passes to the address specified unless the address is equal to 1. If the address is 1, return is made to the next instruction after the "DC AL4(1)" instruction.

3. For the TODACCNT function to be utilized, either the command SET ECMODE ON must have been issued in the virtual machine prior to IPLing CMS, or the EMMODE operand must be specified on the OPTION directory control statement of the issuing virtual machine.

Responses: None

Return Codes:

Register 15 contains one of the following codes.

Return Codes for the ENABLE subfunction:

Code	Meaning
0	ENABLE function successfully completed. The address of the 16-byte area in page zero is returned in the parameter list.
4	ENABLE function has already been issued. The address of the 16-byte area in page zero is returned in the parameter list.
8	ECMODE is not set on.
20	DIAGNOSE 70 has already been issued. CMS is not able to return the timer area address.

Return Codes for the QUERY subfunction:

Code	Meaning
0	QUERY function successfully completed. The 16 bytes of timer information has been transferred from page zero to the parameter list.
12	ENABLE function has not been issued.

Return Codes for the ENABLE and QUERY functions:

Code	Meaning
16	Invalid function specified. Valid functions are 'ENABLE ' or 'QUERY '. This should be an 8-byte field.
24	Function failed because CMS free storage is not available.

WAITRD

Use the WAITRD function to read a line of input from the virtual machine console, the program stack or the terminal input buffer. WAITRD may be executed from an assembler language program via SVC 202 with the following parameter list:

```
PLIST    DS   0F
         DC   CL8'WAITRD'
         DC   AL1(1)
         DC   AL3(input buffer address)
         DC   CL1'code1'
         DC   CL1'code2'
         DC   AL2(length of buffer)
         DC   AL4(prompt buffer address)
         DC   AL4(prompt buffer length)
```

WAITRD first reads from the program stack. If the program stack is empty, WAITRD reads from the terminal input buffer. If the terminal input buffer is empty, WAITRD reads from the virtual machine console. However, if you desire, WAITRD can bypass the contents of the program stack and the terminal input buffer and read directly from the virtual machine console.

After WAITRD reads a line of input, the line is stored in your input buffer. The input buffer address specifies the address of this buffer.

The prompt buffer address and prompt buffer length are optional parameters. If they are used, the prompt information is written from either the buffer specified by the prompt buffer address or your input buffer (if the prompt buffer address is not specified). The prompt buffer length specifies the length of the prompt information to be written prior to the read. Prompt information is written with no carriage return and is used with TTY type devices.

Note: If the prompt parameters are used with code1 = W, Z, *, or $, the read buffer may not be used for the prompt data because the read buffer is cleared prior to the execution of the function.

code1

The following codes specify what kind of processing WAITRD performs on lines read from the terminal input buffer. With these codes you must specify a buffer length of 130 bytes in the 'length of buffer' field in the WAITRD parameter list.

Code	Meaning
U	Reads a logical line, pads it with blanks, and translates it to uppercase.

V	Reads a logical line and translates it to upper case; does not pad with blanks.
S	Reads a logical line and pads it with blanks.
T	Reads a logical line; does not pad with blanks.
X	Reads a physical line.
Y	Reads a logical line, pads with blanks to 130, does no uppercase translation and does not do SET INPUT translation.

The following codes specify what kind of processing WAITRD performs on lines read from the program stack. The length of the input buffer may be up to 255 bytes.

Code	Meaning
W	Reads a physical line; performs no uppercase translation or padding with blanks.
Z	Reads a physical line and translates it to upper case; does not pad with blanks.

Use the following codes when you use APL under CMS. The length of the buffer may be up to 2030 bytes.

Code	Meaning
*	Reads a physical line into the caller's buffer. (See Usage Note 4.)
$	Reads a physical line into the caller's buffer. (See Usage Note 4.)

code2

Code	Meaning
B	Write the prompt information before the read, and read a line of input directly from the virtual machine console.
D	Read a line of input directly from the virtual machine console.
P	Write prompt information before the read.
binary zeros	There is no prompt information, and do not read a line of input directly from the virtual machine console.

The prompt buffer address and the prompt buffer length are specified only if "code2" is B or P.

Usage Notes:

1. Specify the input buffer length as the last parameter in the WAITRD parameter list. Upon completion of the WAITRD function, the 'number of bytes' field contains the number of bytes read.

2. WAITRD does not perform logical line editing when reading a physical line.

   WAITRD performs line editing on lines read from the terminal input buffer (lines typed at the terminal), unless code X is specified; WAITRD does not perform logical line editing when you specify code X. WAITRD does not perform line editing (except uppercase translation, if requested) on lines read from the program stack.

3. Lines typed at the terminal (and stacked in the terminal input buffer) are scanned by CP for logical line editing characters. Logical line editing characters are set by the CP TERMINAL command. The line editing characters may be set for:

   ```
   Chardel
   Linedel
   Linend
   Escape
   ```

   In addition, CMS scans the lines for the following two hexadecimal characters:

   X'00' - interpreted as the end of the physical line. Any character(s) to the right of this hexadecimal character is ignored.

   X'15' - interpreted as the end of the logical line. Any character(s) to the right of this hexadecimal character is interpreted as a new line.

4. For code $, an attention interrupt during a read operation signals the end of the line and does not result in a restart of the read. For code *, an attention interrupt during a read results in a restart of the read operation.

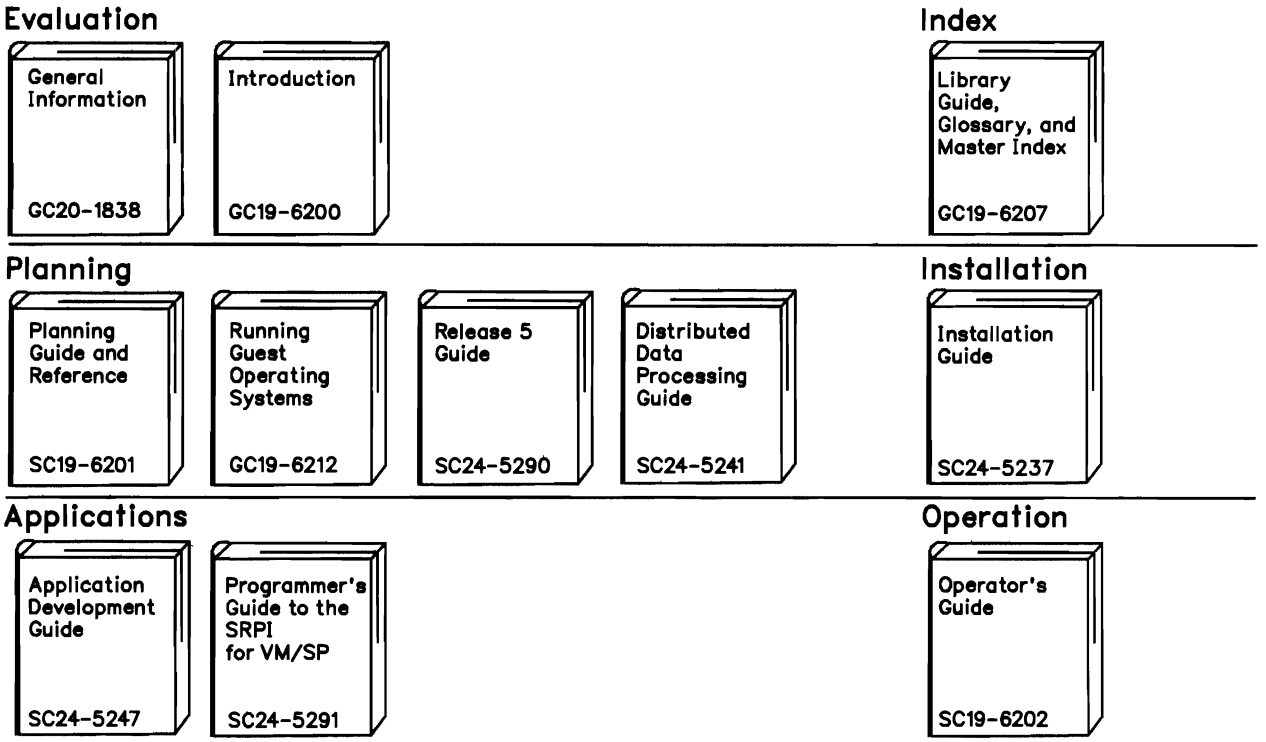
Responses: None

Return Codes:

Code	Meaning
0	Function completed successfully.
2	Invalid code. Read not completed.
4	Code = $. An attention interruption ended the read operation.

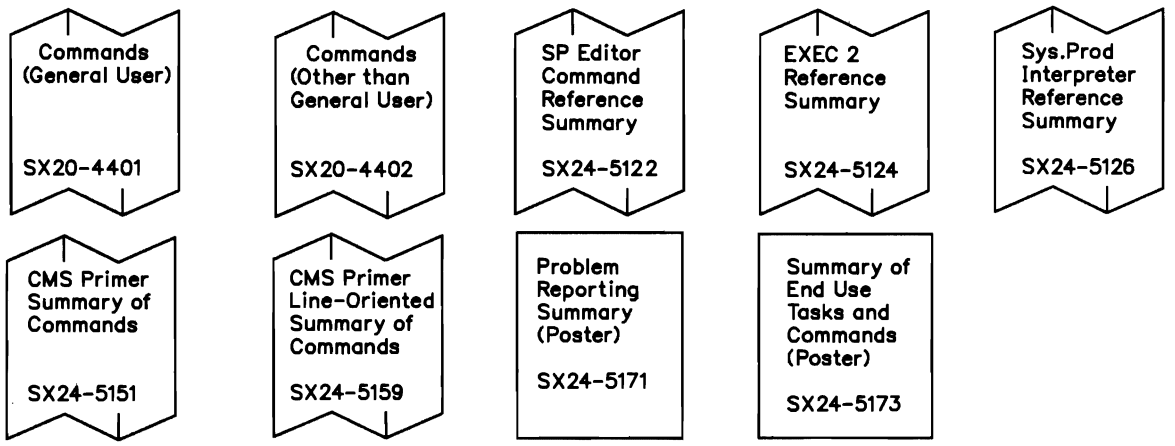
Bibliography

The VM/SP Library (Part 1 of 3)



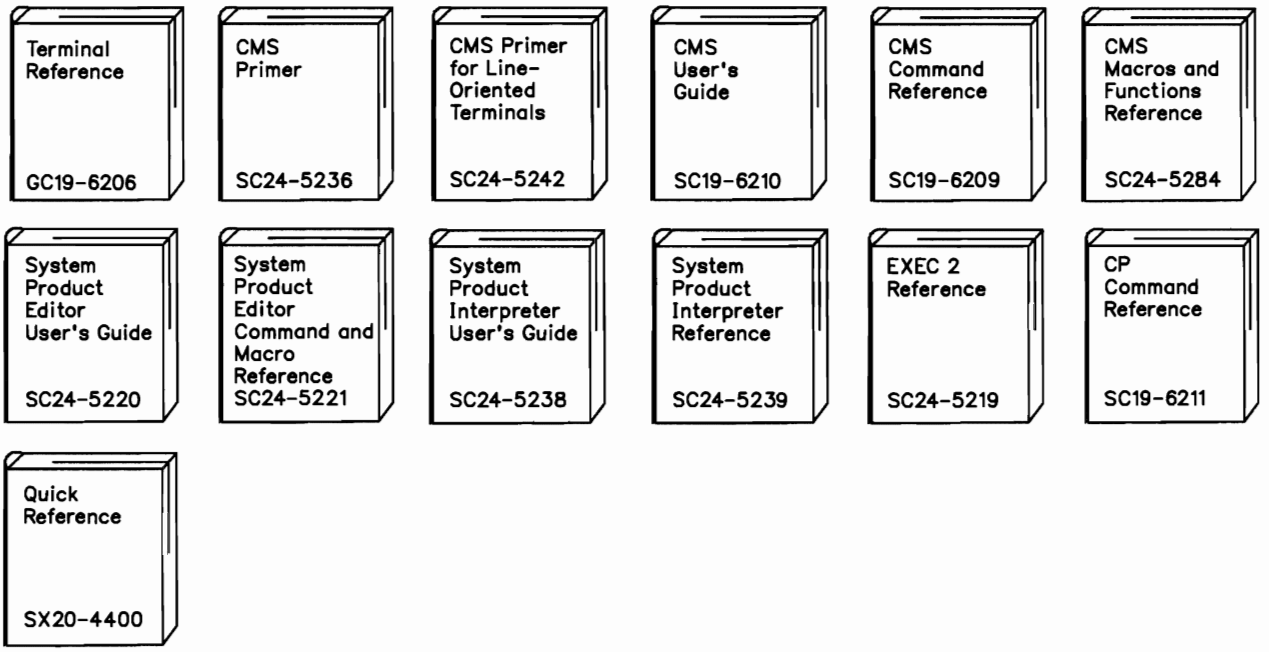
Reference Summaries

To order all of the Reference Summaries, use order number SBOF-3242

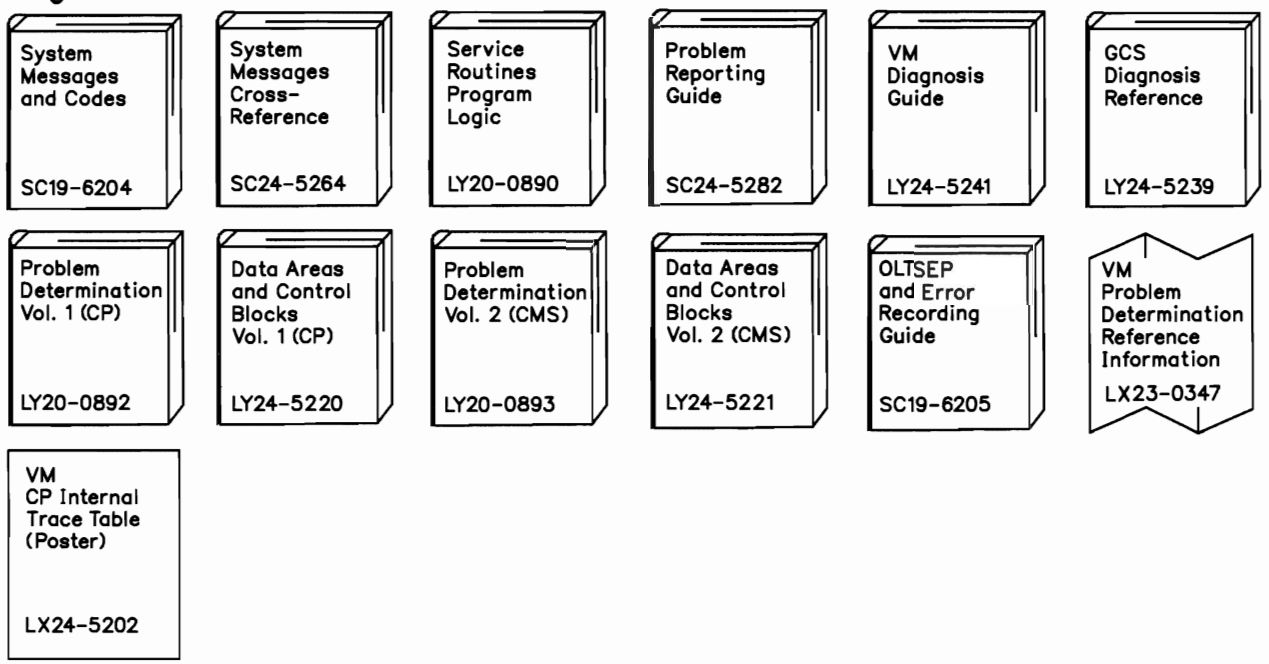


The VM/SP Library (Part 2 of 3)

End Use



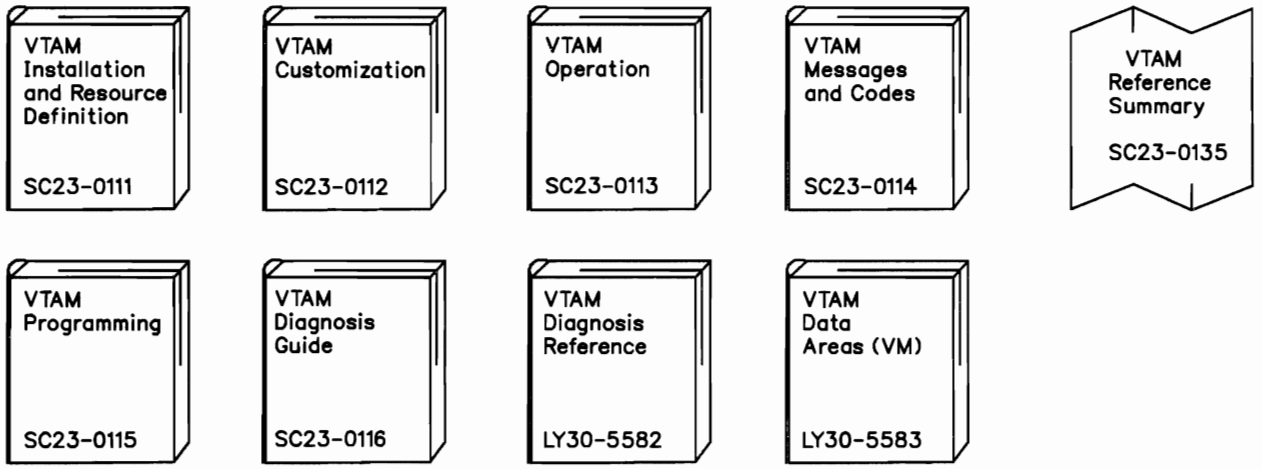
Diagnosis

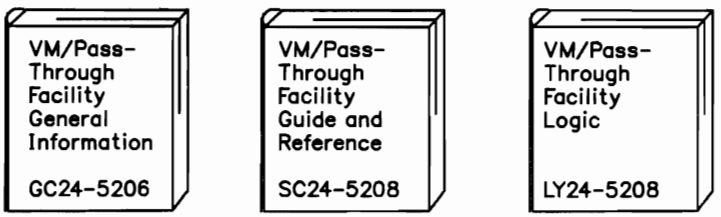


The VM/SP Library (Part 3 of 3)

Administration

Auxiliary Communication Support





Summary of Changes

Summary of Changes
for SC24-5284-0
for VM/SP Release 5

This edition, SC24-5284-0, is a revision of the macros and functions information previously contained in *VM/SP CMS Command and Macro Reference*, SC19-6209-3, and applies to Release 5 of Virtual Machine/System Product (VM/SP).

New CMS Macros for Release 5 of VM/SP

The following CMS macros are new for this release:

ADDENTRY
Places an entry name on a Communications Module termination notification list.

APPLMSG
Retrieves messages from a message repository.

CMSDEV
Returns virtual device characteristics to a specified storage area.

CONSOLE
Accesses CMS full-screen console services.

CPRB
Builds a CPRB DSECT or builds code to acquire storage for and partially initialize a CPRB control block.

CSMRETCD
Defines names for IBM Cooperative Processing return codes for VM/SP.

DELENTRY
Drops entry names previously placed on the Communications Module notification list via the ADDENTRY macro.

LINERD
Reads a line of input from a terminal.

LINEWRT
Displays a line of input at a terminal.

PARSECMD
Parses and translates the arguments of a command.

PARSERCB
Generates a DSECT for the PARSECMD control block.

PARSERUF
Generates a mapping to the Parser Interface for User Token Validation Functions.

PVCENTRY
Generates a DSECT for the Parser Validation Code Table entry.

SENDREQ
Processes service requests.

TRANTBL
Generates a DSECT for the system character set translation tables.

Changed CMS Macros for Release 5 of VM/SP

The following CMS macros have been modified for this release:

PRINTL
The following new operands have been added:

CMSDEV=
Addition of the CMSDEV= operand allows you to specify the type of printer being used.

FORM=
Addition of the FORM= operand allows you to print multiple lines with the execution of a single PRINTL macro.

CC=
Addition of the CC= operand allows you to specify whether or not the data to be printed contains a carriage control character in the first byte of the record.

RDCARD
Addition of the RDAHEAD operand allows you to specify whether or not as many lines as possible are to be read into an internal I/O buffer before each line is read into the specified buffer.

New CMS Functions for Release 5 of VM/SP

The following CMS functions are new for this release:

LANGADD
Adds a language control block to the LANGBLK chain.

LANGFIND
Gets the address of an application's language control block.

Index

A

C

D

E

F

H

I

P

R

S

T

V

W

International Business
Machines Corporation
P.O. Box 6
Endicott, New York 13760

File No. S370/4300-39
Printed in U.S.A.

SC24-5284-0

VM/SP CMS Macros and
Functions Reference
Order No. SC24-5284-0

READER'S COMMENT FORM

Is there anything you especially like or dislike about this book? Feel free to comment on specific errors or omissions, accuracy, organization, or completeness of this book.

If you use this form to comment on the online HELP facility, please copy the top line of the HELP screen.

_____ _____ _____ **Help Information line ____ of ____**

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you, and all such information will be considered nonconfidential.

Note: Do not use this form to report system problems or to request copies of publications. Instead, contact your IBM representative or the IBM branch office serving you.

Would you like a reply? ___YES ___NO

Please print your name, company name, and address:

_____

_____

_____

_____

IBM Branch Office serving you: _____

Thank you for your cooperation. You can either mail this form directly to us or give this form to an IBM representative who will forward it to us.

SC24-5284-0

Reader's Comment Form

CUT
OR
FOLD
ALONG
LINE

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NY

POSTAGE WILL BE PAID BY ADDRESSEE:

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
DEPARTMENT G60
PO BOX 6
ENDICOTT NY 13760-9987

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

VM/SP CMS Macros and
Functions Reference
Order No. SC24-5284-0

READER'S COMMENT FORM

Is there anything you especially like or dislike about this book? Feel free to comment on specific errors or omissions, accuracy, organization, or completeness of this book.

If you use this form to comment on the online HELP facility, please copy the top line of the HELP screen.

_____ _____ _____ **Help Information line ____ of ____**

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you, and all such information will be considered nonconfidential.

Note: Do not use this form to report system problems or to request copies of publications. Instead, contact your IBM representative or the IBM branch office serving you.

Would you like a reply? ___YES ___NO

Please print your name, company name, and address:

_____

_____

_____

_____

IBM Branch Office serving you: _____

Thank you for your cooperation. You can either mail this form directly to us or give this form to an IBM representative who will forward it to us.

SC24-5284-0

Reader's Comment Form

CUT
OR
FOLD
ALONG
LINE

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NY

POSTAGE WILL BE PAID BY ADDRESSEE:

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
DEPARTMENT G60
PO BOX 6
ENDICOTT NY 13760-9987

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

International Business
Machines Corporation
P.O. Box 6
Endicott, New York 13760

File No. S370/4300-39
Printed in U.S.A.

SC24-5284-0

SC24-5284-00

IBM®